Escuela Politécnica Superior

Capsule

Plataforma web para gestionar distintas redes sociales.

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Carlos Aracil Pérez

Tutor/es:
José Vicente Berná Martínez

Junio 2019

Universitat d'Alacant
Universidad de Alicante

Plataforma web para gestionar distintas redes sociales.

Autor:

Carlos Aracil Pérez

Tutor:

José Vicente Berná Martínez.

Departamento: Tecnología informática y computación

Resumen

Castellano

Las redes sociales son un elemento muy importante para que organizaciones, proyectos y pequeñas empresas puedan darse a conocer.

En el siguiente Trabajo de Fin de Grado, se va a realizar una plataforma web desde la cual se podrán gestionar publicaciones en distintas redes sociales, dejar programadas dichas publicaciones para un momento posterior, y obtener unos cuantos consejos con los que conseguir que la publicación sea mejor, más efectiva y llegue a un mayor número de personas. Con esto, mejorará la visibilidad de estas empresas u organizaciones en el mundo de internet y las redes sociales.

Valencià

Les xarxes socials són un element molt important per tal que organitzacions, projectes i xicotetes empreses puguen donar-se a conèixer.

Al següent Treball de Fi de Grau, es va a realitzar una plataforma web des de la qual es podran gestionar publicacions a diferents xarxes socials, deixar programades aquestes publicacions per a un moment posterior, i obtenir uns quants consells amb els que aconseguir que la publicació siga millor, més efectiva i arribe a un major número de persones. Amb açò, millorarà la visibilitat d'aquestes empreses o organitzacions al món d'internet i les xarxes socials.

English

Social networks are a very important element for organizations, projects and small businesses to make themselves known.

In the following Final Degree Project, a web platform will be created from which you can manage publications in different social networks, leave these publications for a future time, and obtain a few tips to make these publications better, more effective and reach a greater number of

people. With this, it will improve the visibility of these companies or organizations in the world of internet and social networks.

Motivación, justificación y objetivo general

La idea de este TFG surgió en la banda de música de la Societat Unió Musical d'Alcoi, a la que pertenezco. Desde hace unos años, soy el encargado de publicar en las redes sociales contenidos y noticias de los diferentes conciertos y eventos que realizamos. Me di cuenta de la cantidad de tiempo que invertía para publicar prácticamente lo mismo en cuatro o más redes sociales distintas: iniciar sesión, escribir la noticia, subir fotos… en cada una de ellas.

Desde antes incluso de empezar este último curso de carrera, siempre he tenido en mente realizar alguna herramienta sencilla para mi uso privado que me permitiese ahorrar el tiempo que se pierde. Quería conseguir algo útil, sin importarme que fuera muy rudimentario, pero después, entre unas cosas y otras, fui dejando de lado esa idea que tenía.

A principios de cuarto curso de carrera, una de las tareas que teníamos que realizar de manera frecuente en una de las asignaturas era subir a diferentes redes sociales noticias y/o actualizaciones del estado en el que se encontraba el proyecto del ABP. Dos de mis compañeras del grupo eran las que principalmente se encargaban de ello, y me di cuenta de una cosa: perdían el tiempo de la misma manera que yo. Entrar a cada red social, de una en una, para acabar poniendo lo mismo en todas.

En ese momento, recordé aquel proyecto que siempre quise hacer, pero nunca realicé, y es por esto por lo que decidí llevar a cabo este proyecto para mi TFG.

Esta herramienta~~,~~ consistirá en una plataforma web, que se llamará **Capsule**. El porqué de este nombre es por el simple hecho de que "encapsulará" varias redes sociales en el mismo sitio. Además, viene también por el hecho de que el estrés y la falta de tiempo en el trabajo ya se considera como "la enfermedad del siglo XXI". Si quieres publicar lo mismo en varias redes sociales, ¿por qué perder el tiempo entrando de una en una cuando puedes ahorrarte gran parte de ese tiempo haciéndolo desde un mismo sitio? Gracias a esta "cápsula" quizá ganes algo más de tiempo para realizar otras tareas.

Desde **Capsule**, el usuario podrá añadir distintas redes sociales para que se sincronicen. Podrá gestionar distintas opciones individuales de cada una de ellas, pero el contenido básico y más importante, estará en la pantalla principal de la aplicación, desde donde podrá escribir la publicación y que se publique en aquellas que el usuario decida y que previamente haya sincronizado con Capsule. Otra de las opciones que se podrán realizar y que es interesante ya

que también ayuda al ahorro de tiempo en el uso de las redes sociales, será la de poder programar las publicaciones. No será necesario buscar tiempo para publicar en la hora determinada que te interese, Capsule lo hará por ti. Para todo esto y también para que el usuario de mi aplicación no olvide nada, dispondrá de un sistema de notas, recordatorios o similar con opción para notificaciones. Además, al igual que puedes publicar lo mismo en varios sitios a la vez, también podrás ver las estadísticas de cada publicación de cada red social desde un mismo sitio.

También, al tratarse de una plataforma web, el usuario no tiene que preocuparse por perder alguno de sus dispositivos, quedarse sin batería o un largo etcétera de problemas que le puedan suceder. Todo se encontrará en el servidor de Capsule, y desde cualquier dispositivo con acceso a internet podrá entrar y utilizar sus redes sociales.

Por todo lo comentado anteriormente, creo que es un tema interesante como TFG y a la vez viable. Además, es bastante útil para todo aquel que gestiona distintas redes sociales de alguna pequeña sociedad u organización como es mi caso, y también para aquellos que lo hacen por gusto o simplemente para dar a conocer un pequeño negocio, y que suelen publicar el mismo contenido en cada una de ellas.

Desconozco si la aplicación acabará siendo utilizada por la gente o simplemente será un TFG más que acabará en el olvido. Por un lado, dependerá del éxito que pueda tener en el mercado. No es la primera aplicación que existe que gestiona diversas redes sociales desde un mismo sitio. Sin embargo, ésta está enfocada para quienes buscan algo sencillo y gratuito. Gestionar la difusión de ABP, llevar distintas redes de algún club local, intentar promocionar un pequeño negocio o la imagen personal de alguien… Las aplicaciones que actualmente se encuentran en el mercado son de pago y disponen de varias opciones enfocadas para empresas más grandes y conocidas, como ayuda y consejos para el SEO, análisis de la competencia o “geofocalizar” distintos contenidos dependiendo del país al que van dirigidos. Todo esto puede echar hacia atrás a un gran grupo de usuarios, como por ejemplo mi grupo del ABP, un grupo de teatro, un profesor, un entusiasta de la fotografía que quiere publicar una foto bonita que ha hecho, o sin ir más lejos, yo también en mi banda de música, donde publicamos algún que otro contenido en internet.

Hablando de promocionar cosas en redes sociales, yo mismo con este TFG seríamos también un potencial cliente de **Capsule**. Creo que, con una buena difusión, esta plataforma web puede

acabar dándose a conocer y siendo útil para diferentes colectivos (y, si esto pasa, me sentiría más que satisfecho).

Aun así, creo que lo más importante no es si servirá, o si tendrá éxito o no. Es un trabajo que me motiva, algo que siempre he querido hacer por gusto y de paso algo donde poder aprender a utilizar las APIs de distintas redes sociales, así como otras herramientas o protocolos como OAuth2. Servirá para mí, para aprender cosas nuevas que no hemos visto en la carrera y para sentirme satisfecho conmigo de haber hecho algo que me gusta.

Cuando termine el trabajo, creo que demostrará mis habilidades como Ingeniero Multimedia que ha elegido el itinerario de Gestión de Contenidos, puesto que para poder llevarlo a cabo tendré que trabajar tanto el backend como el frontend, demostrando algunos de mis conocimientos y aprendiendo tecnologías nuevas que sé que en un futuro me pueden ser de gran ayuda.

Agradecimientos

Después de cuatro años de estudios de grado, y de más de seis meses de preparación de este Trabajo de Fin de Grado, quiero aprovechar para agradecer el apoyo recibido de todas aquellas personas que han estado a mi lado durante este proceso.

En primer lugar, quiero darles las gracias a mis padres, por haberme dado la libertad de optar por el grado que libremente quise elegir; por haberme inculcado, desde siempre, la constancia, el no tirar la toalla al primer intento, el dar una segunda oportunidad a las desilusiones, el saber mirar hacia la meta y hacia el futuro; y por saber que siempre, siempre, independientemente de los resultados, están aquí para apoyarme.

En segundo lugar, quiero agradecer a mis compañeros del ABP y de otros trabajos en equipo a lo largo de toda la titulación: Dome, Alberto, Blanca y Natalia por el constante apoyo a lo largo de todo este camino, tanto a nivel académico compartiendo con ellos la motivación, los conocimientos y los distintos puntos de vista en cada proyecto; como a nivel personal, por haberme hecho sentir parte de un grupo, compartiendo juntos momentos difíciles y, sobre todo, horas de charlas, bromas y risas.

Por último, quiero también agradecer a mis abuelos y a mi hermana, por todo el cariño y el respaldo que siempre me dan en cada una de las cosas que emprendo; y cómo no, mi agradecimiento también va para el resto de compañeros geniales con los que he coincidido en la carrera como Alexei, Mark, Juanan, Jesús, Martín, Jaime, David (el cifu), Mer y Yolanda; que, junto a mis amigos y compañeros del ABP-mencionados anteriormente, han conseguido que mi paso por esta Universidad haya sido una experiencia inolvidable.

¡Muchas gracias a todos!

Citas

“If you think you are too old to rock 'n roll, then you are.”

Lemmy Kilmister

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Este Trabajo de Fin de Grado consiste en una aplicación o plataforma web llamada **Capsule** que permite a los usuarios no profesionales que necesiten administrar varias redes sociales con una mínima inversión temporal, la posibilidad de hacerlo sin tener que recurrir a aplicaciones más complejas.

Capsule está dirigida, por tanto, a ese segmento de clientes que, sin tener excesivos conocimientos y con pocos recursos temporales y económicos, desee gestionar, desde un mismo sitio y de una manera intuitiva y ágil, distintas redes sociales (Facebook, Twitter, Instagram y Linkedin) con un pequeño seguimiento posterior. Esta plataforma les ofrecerá la oportunidad de poder entrar y distribuir contenido en todas sus redes de una forma asistida, económica, rápida y sencilla.

¿Por qué estas redes sociales? Porque todas ellas se encuentran entre las más utilizadas en 2018 [1] y a todas se le da un uso similar: publicar texto junto con alguna imagen.

Hoy en día, las redes sociales son una herramienta imprescindible para difundir contenido de todo tipo. Cualquier empresa, por pequeña que sea, así como cualquier asociación o colectivo puede utilizarlas para darse a conocer, atraer a más clientes y, por ejemplo, mantener la comunicación con los mismos, resolviendo las dudas que puedan plantear en cualquier momento y a cualquier hora del día.

Hay profesionales que se dedican explícitamente a ello y, en el mercado, tienen a su disposición multitud de aplicaciones que les pueden facilitar dicha gestión de las redes sociales y que incorporan una gran cantidad de opciones para optimizar al máximo el rendimiento; pero también encontramos otro tipo de personas que realizan esas mismas tareas pero a un nivel más sencillo y no profesional y que, por lo tanto, no necesitan hacer uso de herramientas complicadas, sofisticadas y tan profesionales, ni les sale rentable pagarlas, ni, en algunos casos, tienen los suficientes conocimientos informáticos para poderlas utilizar.

El signo distintivo de Capsule es la sencillez de manejo y la rapidez a la hora de publicar una misma entrada o post, en las distintas redes sociales que maneje el cliente, evitando la pérdida de tiempo al replicar el contenido en cada una de ellas por separado. Esta herramienta facilitará la vida a todo el colectivo de usuarios que no realizan una actividad profunda de marketing ni buscan conseguir el máximo rendimiento posible; pero sí hacen algunas tareas de “Community

Manager", como las de replicar un mismo contenido en varias redes sociales a la vez y comprobar qué opiniones o reacciones ha tenido por parte de los usuarios. No necesita analíticas, seguimiento, SEO, marketing… pero sí algo fácil que les ahorre tiempo y sea sencillo de utilizar.

Para que resulte aún más útil y eficaz a los usuarios, la plataforma incluirá, también, la posibilidad de programar entradas en las distintas redes sociales, en horas y días concretos; pudiendo dejar una publicación preparada y poder centrarse así en otras cosas. Además, se añadirá una sección donde el cliente podrá realizar anotaciones y elegir si quiere que la aplicación le recuerde dicha anotación (a modo de notificación).

Teniendo en cuenta que el posible usuario potencial de Capsule puede no disponer de muchos conocimientos sobre cómo utilizar y publicar contenido en las redes sociales, también se mostrarán sugerencias, de manera que le puedan ayudar a crear publicaciones de mayor calidad y que lleguen a un mayor número de usuarios.

A la hora de limitar el alcance del producto, he decidido empezar implementando cuatro redes sociales: Facebook, Twitter e Instagram (que son las tres redes sociales más utilizadas en la actualidad), añadiendo Linkedin (como red social más enfocada a negocios). No he incluido otras redes sociales basadas principalmente en vídeos, como Youtube, puesto que el concepto de uso es diferente al de las anteriores, centradas más en difundir texto y, opcionalmente, imágenes. De todas formas, dependiendo de cómo avance el TFG, podría añadir en un futuro más redes sociales e implementar otras funcionalidades para la aplicación.

2. Estudio de viabilidad

En este apartado voy a analizar el proyecto en sí mismo, sus pretensiones y objetivos haciendo uso de la herramienta Lean Canvas. Se trata de un lienzo de modelo de negocio que nos ayuda a diseñarlo de forma visual. Analiza un proyecto desde diferentes perspectivas: por un lado, tenemos el mercado y, por otro, el del proyecto; separándolo en diferentes bloques con distintas perspectivas de interés:

Segmento de clientes:

En general, el servicio que ofrece la aplicación iría dirigido a cualquier tipo de usuario que necesite manejar múltiples redes sociales, de una forma sencilla, con poca inversión de tiempo y sin necesidad de conocimientos complejos.

Podría concretar en un cliente potencial de perfil no profesional, que no puede invertir tiempo en aplicaciones más complejas ni en herramientas sofisticadas; simplemente necesita una plataforma gratuita para él, de uso fácil que le permita distribuir sus contenidos en las diversas redes sociales que maneja, y poder obtener un pequeño seguimiento sobre lo que ha ocurrido en ellas.

Como ejemplo dentro de los posibles usuarios a los que les resultaría útil este servicio, citaría a las asociaciones vecinales, grupos juveniles, clubes deportivos, pequeñas o medianas empresas, etc., que utilizan sus redes sociales con el fin principal de darse a conocer, informar, captar nuevos socios o clientes y mantener los que tiene.

Problemas:

Teniendo en cuenta el segmento de posibles clientes, que se corresponde con un perfil de usuario no profesional, que suele realizar la función de Community Manager de una forma desinteresada y fuera de horas de trabajo, los principales problemas encontrados son:

- Para publicar el mismo contenido en las tres o cuatro redes principales que maneja debe invertir mucho tiempo.
- Para poder ver las reacciones de los usuarios a una misma publicación en distintas redes sociales, las únicas opciones con las que cuenta son acceder a cada red social por separado.
- Dado que no es su trabajo principal, puede olvidar o retrasar la publicación del contenido.
- El potencial cliente no tiene excesivos conocimientos informáticos ni de manejo de redes sociales.

- No cuenta con un presupuesto económico para el apartado de comunicación en redes sociales.
- Las herramientas existentes en la actualidad y que le resultaría de utilidad para no tener que replicar las entradas en cada red social, son complejas de uso y ofrecen muchas más funciones que las necesarias para ese perfil de usuarios al que va dirigida, además, conllevan una inversión económica.

Proposición única de valor:

Servicio capaz de ofrecer al usuario "amateur", de una forma ágil, sencilla y guiada, la posibilidad de publicar y administrar a la vez distintas cuentas en redes sociales, donde replicar una misma entrada en todas ellas, añadiendo además, opciones como la posibilidad de programar publicaciones para el futuro y recibir alertas (bien por correo electrónico o mediante las notificaciones del navegador web); así como también la gestión de notas para que el usuario no olvide nada.

Solución:

A los problemas descritos anteriormente:

- Ofrecer un servicio de fácil manejo, ágil y guiado que permita publicar la misma entrada a la vez en distintas redes sociales.
- Posibilidad de gestionar las distintas redes sociales, agruparlas y realizar las publicaciones de todas ellas desde un mismo lugar, viendo, además, las reacciones que han tenido los usuarios de cada red social a una misma publicación.
- Posibilidad de programar una publicación.
- Ayudas y sugerencias para que la publicación de un usuario que no tiene ni demasiados conocimientos informáticos ni entiende bien las redes sociales, pueda llegar a un mayor número de personas.
- Posibilidad de realizar anotaciones y avisos. Estos recordatorios podrán ser a través de correo electrónico al usuario, o a través de una notificación en el navegador web.

Canales:

Dar a conocer la aplicación mediante una web corporativa y a través de RRSS como Twitter, Facebook e Instagram.

Flujo de ingresos:

El servicio será gratuito para el usuario. Estará habilitado un servicio de donaciones mediante Paypal para todo aquel que desee aportar algo de dinero, pero se financiará, principalmente, mediante la inserción de anuncios.

Estructura de costes:

- Dominio web (12€ al año).
- Servidor VPS (10€ al mes, para algo sencillo, pero a la vez adecuado).
- Tiempo estimado de 300 horas aproximadamente (40€/hora, 12.000€).

Métricas clave:

- Cantidad de usuarios registrados; para saber cuántos clientes en total tengo.
- Cantidad de usuarios registrados que han publicado algo en el último mes; para saber cuántos clientes en activo tengo.
- Cantidad de posts enviados a través de este servicio en el último mes; para saber el uso real que tiene esta aplicación.
- Número de visitas a la página web; para saber si está funcionando el plan de difusión.

Ventaja diferencial:

Servicio único que, de una forma ágil, sencilla y gratuita, permite ahorrar tiempo a la hora de publicar un mismo contenido en diversas redes sociales.

2.1. Lean Canvas

Problemas:	**Solución**:	**Proposición de valor única:**	**Ventaja diferencial:**	**Segmento de clientes:**
Para publicar el mismo contenido en las tres o cuatro redes principales que maneja debe invertir mucho tiempo. Para poder ver las reacciones de los usuarios a una misma publicación en distintas redes sociales, las únicas opciones con las que cuenta son acceder a cada red social por separado. Dado que no es su trabajo principal, puede olvidar o retrasar la publicación del contenido. El potencial cliente no tiene excesivos conocimientos informáticos ni de	Ofrecer un servicio de fácil manejo, ágil, gratuito y guiado que permita publicar la misma entrada a la vez en distintas redes sociales. Posibilidad de gestionar las distintas redes sociales, agruparlas y realizar las publicaciones de todas ellas desde un mismo lugar, viendo, además, las reacciones que han tenido los usuarios de cada red social a una misma publicación. Posibilidad de realizar anotaciones y avisos. Estos recordatorios podrán ser a través de correo electrónico al usuario, o a través de una notificación en el navegador web.	Servicio capaz de ofrecer al usuario, de una forma ágil y sencilla, la posibilidad de publicar y administrar a la vez distintas cuentas en redes sociales, donde replicar una misma entrada en todas ellas, añadiendo la opción de programar tareas y recibir alertas (bien por correo electrónico o mediante las notificaciones del navegador web); así como también la gestión de notas para que el usuario no olvide nada.	Servicio único que, de una forma ágil, sencilla y gratuita, permite ahorrar tiempo a la hora de publicar un mismo contenido en diversas redes sociales.	Cualquier tipo de usuario que necesite manejar múltiples redes sociales, de una forma sencilla, con poca inversión de tiempo y sin necesidad de conocimientos complejos.

<table>
<tr><td>manejo de redes sociales.

No cuenta con un presupuesto económico para el apartado de comunicación en redes sociales.

Las herramientas existentes en la actualidad y que le resultaría de utilidad para no tener que replicar las entradas en cada red social, son complejas de uso, ofrecen muchas más funciones que las necesarias y conllevan una inversión económica.</td><td>Métricas clave:

Cantidad de usuarios registrados, usuarios registrados que han publicado en el último mes, posts enviados a través de la aplicación en el último mes y número de visitas a la web corporativa.</td><td></td><td>Canales:

Dar a conocer la aplicación mediante una web corporativa y a través de RRSS como Twitter, Facebook e Instagram.</td><td></td></tr>
<tr><td colspan="3">Estructura de costes:

Dominio web, servidores y tiempo.</td><td colspan="2">Flujo de ingresos:

Donaciones mediante PayPal y anuncios.</td></tr>
</table>

Tabla 1. Cuadro para el análisis Lean Canvas
(Fuente propia)

3 Análisis de riesgos

En este apartado pretendo reflejar todos los problemas y, por tanto, riesgos que puede sufrir el TFG a lo largo de todas sus fases de desarrollo, cómo se pueden prevenir, minimizar los daños y elaborar un plan de contingencia ante ellos.

3. Tipo de riesgo	Riesgo	Estrategia
Tecnológico.	Problema con alguna API de alguna red social.	**Prevención**: Estudiar las API antes de incorporarlas. **Minimización:** Utilizar alguna API de terceros que solvente el problema. **Plan de contingencia:** Eliminar dicha red social de la aplicación o sustituirla por una red social similar.
Tecnológico.	Pérdida del trabajo o de las últimas modificaciones.	**Prevención:** Tener copias de seguridad en algún dispositivo distinto del portátil donde trabajo, o en servicios en la nube (Github para código, Google Drive para documentación; o similares).
Tecnológico.	Pérdida de acceso al servicio (cuando esté en un servidor web, para poder acceder desde cualquier dispositivo con internet); bien por un fallo en la configuración o por un ataque.	**Prevención:** Tener configuradas copias de seguridad tanto de proyecto como de la configuración del servidor, así como una buena seguridad en el servidor para evitar ataques. **Plan de contingencia:** Restaurar las copias de seguridad en otro servidor.
Personal.	Falta de tiempo.	**Prevención:** Llevar el TFG con un buen ritmo de trabajo. **Plan de contingencia:** Eliminar alguna de las 4 redes sociales, y dejarlo en 3 o centrarse en la funcionalidad principal y eliminar funcionalidades extra.
Personal.	Dejadez en el TFG	**Prevención:** Reuniones periódicas con el tutor. **Minimización:** Dedicar un mayor esfuerzo para ponerse al día.

Herramientas.	Desconocimiento en la utilización de las herramientas usadas en el proyecto.	**Prevención:** Informarse de la curva de aprendizaje que requieran las herramientas, viendo si son necesarias para el proyecto y aprender a utilizarlas **Minimización:** Buscar información o preguntar a compañeros para que le ayuden a utilizarla. **Plan de Contingencia:** Cambiar de herramienta a una conocida o preguntar al profesor o un experto sobre su uso.

Tabla 2. Análisis de riesgos.

4. Planificación

Una buena planificación desde el principio puede ayudar en la consecución del proyecto. Además, me permitirá una vez terminado el proyecto analizar, junto a los resultados, la adecuación de la planificación al desarrollo completo del proyecto.

Contenidos	Tiempo total	Fecha límite fin
Motivación, justificación, objetivo general, Introducción Estado del arte	1 mes	27 diciembre
Objetivos Metodología Análisis y especificación Presupuesto, estimaciones, planificación	1 mes	25 enero
Diseño	2 meses	10 marzo
Implementación	1 mes	30 abril
Pruebas y validación Resultados Conclusiones y trabajo futuro Referencias, bibliografía y apéndices Agradecimientos, citas, índices	1 mes	1 junio

Tabla 3. Planificación temporal TFG

5. Estado del arte.

5.1. Estudio de la problemática.

¿Quién no se ha visto obligado alguna vez a realizar algunas tareas como si fuera un Community Manager? En mi propio entorno tengo varios ejemplos, empezando por mí, en la agrupación musical a la que pertenezco; o por mis padres, cuando formaban parte del AMPA y del club deportivo del colegio e instituto donde estudié.

Muchas personas necesitan, diariamente, gestionar y publicitar en redes sociales desde su propio negocio hasta, por ejemplo, las actividades que realiza la entidad, asociación o grupo al que puedan pertenecer.

Hoy en día esta situación es la habitual, el 85% de los españoles utiliza las redes sociales. La media nos dice que tenemos casi 5 redes sociales distintas cada uno [1]. No es de extrañar, por tanto, que la mayoría de las empresas y organizaciones utilicen varias cuentas en redes sociales para intentar poder llegar a la mayor cantidad de usuarios posibles.

Sin embargo, no vale sólo con estar en redes sociales, hay que estar al día y mantener un flujo constante de comunicación [2]. Para ello, es necesario publicar contenido, y no sólo en una red social. El público objetivo se encuentra en varias redes sociales y, aunque este público maneje distintas redes sociales, no todos las utilizan con la misma frecuencia. El uso de las redes sociales depende también de la edad. Una misma red social no se utiliza con la misma frecuencia entre la gente joven y la gente adulta. Dejando WhatsApp de lado, puesto que, aunque es una red social, se trata de un chat y no entra dentro del tipo de red social que se busca para este estudio, Facebook es la red social con más usuarios en España. A pesar de esto, la gente joven a diario prácticamente no la utiliza, mientras que para los mayores de 45 años es la red social preferente [1].

Por este motivo, las empresas y organizaciones acaban, como mínimo, con un par de redes sociales. Ya en 2014, el 85% de las empresas españolas utilizaba redes sociales. Las más utilizadas son: Facebook, Twitter, Instagram y Linkedin [3].

Además, dependiendo del tipo de contenido que quieran publicar, es posible que una publicación sólo se quiera en unas redes sociales en concreto, o quizá se quiere publicar en todas excepto en una. Por este motivo, Capsule permite publicar un mismo post en distintas redes

sociales, seleccionando siempre a cuáles de ellas va dirigido. Una publicación dirigida a gente joven quizá sólo se quiere publicar en Twitter e Instagram, mientras que otra se prefiera dirigir a todo tipo de público y se quiera publicar en todas las redes sociales.

5.2. Necesidad de programar tareas

Atendiendo al Estudio Anual sobre Redes Sociales en el año 2018 [1], podemos observar que el mayor momento del día en que los españoles hacemos uso de las redes sociales es por la noche, entre las 20:30 y las 00:30.

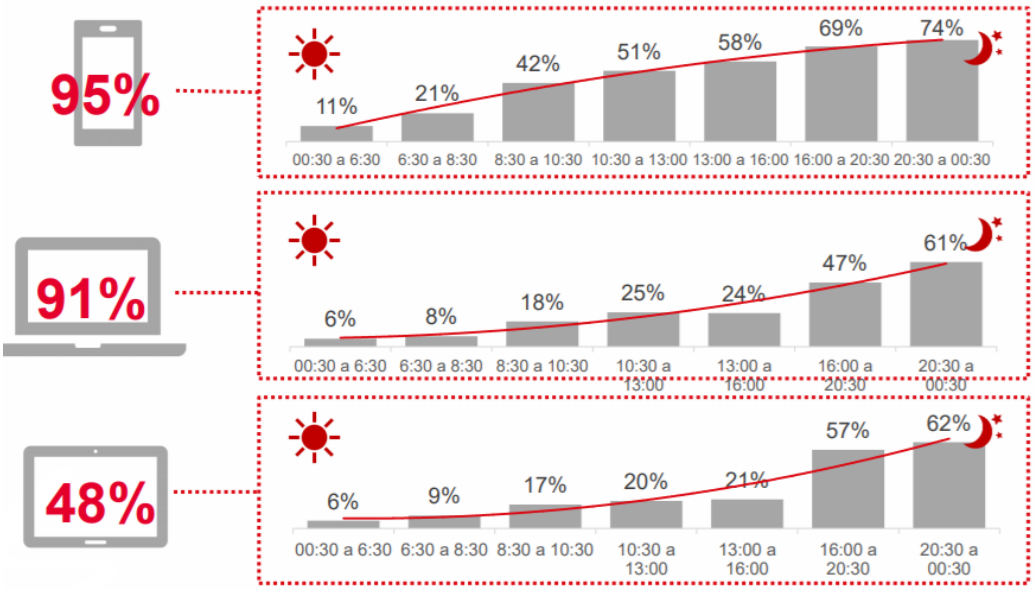


Figura 1. Horarios de conexión en España (https://iabspain.es/wp-content/uploads/estudio-redes-sociales-2018_vreducida.pdf)

Este dato coincide, prácticamente, con el horario de finalización de la jornada laboral de la mayoría de los españoles [5] [6]. Es también el momento en el que existe una mayor saturación de internet por parte de los proveedores [7].

Esto, es bastante entendible, puesto que es el momento en el que la gente llega a casa, después de todo un día de trabajo, para descansar y relajarse. Gran parte de esta actividad pasa por acceder a las redes sociales.

Por este motivo, la tarde-noche sería el momento del día idóneo para lograr que tu publicación en las redes sociales sea vista por el mayor número de usuarios posible.

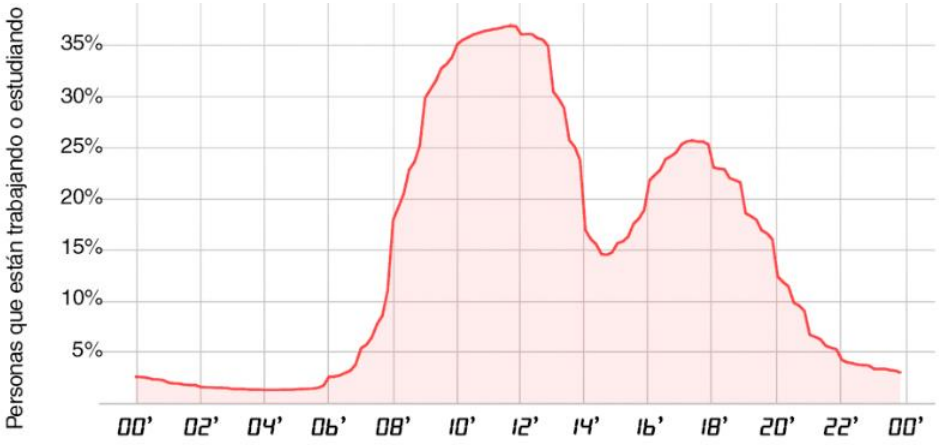


Figura 2. Estadística del horario laboral elaborada por Eurostat (https://www.europapress.es/sociedad/noticia-asi-compara-horario-laboral-espanol-otros-paises-europeos-20161213135213.html)

Si tenemos en cuenta el segmento de clientes de esta plataforma web, que como hemos indicado se trata de gente "amateur", el publicar justamente en ese momento del día, puede ser uno de los motivos que te haga desistir del intento. Recordemos que este grupo de clientes, que hace la función de "Community manager" de manera no profesional, deben buscar un hueco en su tiempo libre, después de su propio trabajo, para promocionar su empresa, agrupación o colectivo al que pertenecen, por lo que muchas de estas publicaciones acaban subiéndose tarde o con retraso y en un horario en el que la mayoría del público no se encuentra conectada.

5.3. Cuatro redes sociales

Facebook, Twitter, Instagram y Linkedin. Según el estudio anual de redes sociales en España [1], los españoles tienen de media entre 4 y 5 redes sociales por cabeza. Entre las 4 redes más utilizadas encontramos Facebook, Whatsapp, Youtube, Instagram. Obviamente, para esta plataforma web, redes sociales como Whatsapp, que es un chat, o plataformas de vídeos como Youtube no encajan en el tipo de red social buscada. Capsule se va a enfocar en aquellas pensadas en publicar texto + alguna imagen.

Teniendo en cuenta este tipo de redes sociales, las cuatro más populares que encontramos son Facebook, Instagram, Twitter y Linkedin [1]. Así, alguien que busca que su contenido llegue al mayor número de personas, buscará que su contenido se pueda publicar en cualquiera de estas

cuatro redes sociales, y el usuario de Capsule podrá escoger cuál de ellas desea vincular en la aplicación.

5.4. Principales competidores

Después de realizar un estudio mediante la búsqueda de las principales aplicaciones de gestión de múltiples redes sociales, lo primero que llama la atención es que no son pocas; hay muchísimas aplicaciones o plataformas web que realizan este tipo de tareas. Sin embargo, no hay ninguna que destaque por encima de las demás; si buscamos artículos sobre cuál de todas es la mejor, dependiendo de quién redacte el artículo, nos recomendará una u otra [9, 10]. Aun así, en la mayoría de los casos, encontramos algunas siempre entre las más recomendadas o entre las más conocidas. La principal de ellas, a fecha de diciembre de 2018, es Hootsuite.

A continuación, voy a analizar cómo funcionan estas herramientas, cómo es el ciclo de creación de contenidos, cómo se publica y cómo se ve la información y se controla; para tener en cuenta todo esto a la hora de realizar mi propuesta.

5.4.1. Hootsuite

Permite registrarse con email o contraseña o haciendo uso de la cuenta de Google, Twitter o Facebook del usuario.

En la pantalla principal, aparecen las publicaciones realizadas. En la parte superior, hay una pequeña barra con un botón para crear una nueva publicación. Hay también un menú en el lateral izquierdo, donde se encuentran las diferentes herramientas avanzadas que utiliza Hootsuite.

A la hora de publicar, te permite elegir a qué redes sociales va a ir dirigido el contenido. Si no tienes ninguna red añadida aún, te aparece una sección para hacerlo. Bajo de esto, permite seleccionar las imágenes o vídeos, y finalmente el campo de texto.

Durante todo este proceso, en el lateral derecho, aparece una previsualización de cómo quedará la publicación en cada red social.

Tiene un precio que empieza en los 19,90€ al mes, y ofrece un plan gratuito para gestionar únicamente 3 redes sociales.

Entre las empresas que utilizan Hootsuite encontramos Mapfre, Meliá Hoteles, Orange o 1&1, entre otros.

5.4.2. Sendible

No ofrece el poder registrarse haciendo uso de Google, Facebook u otra red social. Hay que hacerlo sí o sí mediante un correo electrónico y una contraseña.

Una vez dentro, vemos que es bastante similar. La pantalla principal, en lugar de mostrar las últimas publicaciones realizadas, muestra los comentarios y mensajes que otros usuarios han realizado hacia tus cuentas de redes sociales. También encontramos una barra superior, con un botón para crear una nueva publicación. Sin embargo, las herramientas avanzadas para empresas; así como las redes sociales que tienes conectadas, también se encuentran en esta barra superior.

En la barra del menú lateral, encontramos las opciones para programar publicaciones, calendario y una gestión de tareas.

A la hora de redactar una publicación, primero se elige con qué redes sociales, luego se redacta el texto, y luego se eligen las imágenes. Al final, se encuentran dos botones, uno para enviar la publicación y el otro para programarla.

Sendible fue diseñado principalmente con la idea de administrar múltiples actividades de marketing y aumentar las ventas, y por este motivo, las herramientas específicas para empresas o usuarios más avanzados se encuentran en la barra superior.

No ofrece ningún plan gratuito, y sus planes empiezan en los 20€ al mes.

Entre las empresas que utilizan Sendible encontramos Toyota, Sony Music o la BBC, entre otros.

5.4.3. Buffer

Permite iniciar sesión con Twitter, Facebook y Linkedin. La primera vez que se entra en la aplicación, te obliga a seleccionar qué redes sociales vas a conectar (se pueden modificar más adelante).

Una vez dentro, la pantalla principal consiste en un campo de texto, donde empezar a escribir; y en la barra izquierda aparecen tus redes sociales conectadas. Simplemente hay que hacer click en qué redes sociales quieres publicar, y luego empezar a escribir la publicación.

Encontramos dos barras superiores. Una con la configuración de la cuenta de usuario, y otra bajo de esta con las opciones para programar posts, ver las reacciones que han tenido otros posts tuyos, y un desplegable con más herramientas.

Tampoco tiene versión gratuita, y las tarifas de pago empiezan en los 39€ al mes.

5.4.4. Otras opciones

En todas las anteriormente comentadas, encontramos un factor común: ofrecen una gran cantidad de herramientas enfocadas a promocionar una gran empresa a nivel mundial, como el análisis SEO, reportar o moderar comentarios automáticamente, creación de imágenes desde la plataforma o incluso el poder trabajar en equipo.

Sin embargo, el segmento de clientes al que está enfocado mi servicio, no busca casi ninguna de estas opciones. Buscan algo más sencillo y fácil, y además gratuito, puesto que la mayoría de las existentes son de pago. Si buscamos alternativas 100% gratuitas a estas aplicaciones, únicamente encontramos una: Alternion, que permite gestionar varias redes sociales y cuentas de correo electrónico desde un mismo sitio de manera totalmente gratuita, no obstante, la última actualización de dicha herramienta se realizó en octubre de 2017 y se encuentra actualmente descontinuada.

5.5. Conclusiones

Tras analizar las anteriormente comentadas, encuentro cosas bastante útiles en todas ellas, y también otras que, pensando en mi segmento de clientes, no serían necesarias en mi herramienta.

Iniciar sesión mediante alguna red social, o con una cuenta de correo electrónico existente es algo que facilita bastante el empezar a utilizar el servicio, ya que no necesitas rellenar ningún formulario de registro, ni necesitas recordar más contraseñas. Por tanto, a la hora de desarrollar mi herramienta, incluiré mínimo dos servicios (en un principio, Google y Facebook, puesto que son el servicio de correo más utilizado en el mundo [13] y la red social más utilizada en España [1] y, así, poder hacer que la mayoría de usuarios puedan iniciar sesión sin necesidad de registrarse).

Teniendo en cuenta que mi herramienta va enfocada a personas con pocos conocimientos sobre redes sociales, tras analizar las otras herramientas similares, se ha decidido un estilo similar al de Buffer: que la pantalla principal muestre directamente el campo para empezar a redactar publicaciones, y evitar así hacer un click adicional en un botón de una barra superior. Teniendo en cuenta que ésta es la funcionalidad principal de la herramienta, es más fácil e intuitivo

encontrarla directamente nada más iniciar sesión, sin necesidad de hacer clicks en barras o menús.

Sin olvidar que el segmento de clientes al que va dirigida la aplicación no es experto, poder ver cómo quedará la publicación antes de que el usuario la envíe, tal como hace Hootsuite, es algo muy útil para no cometer errores al publicar, por lo que, será una funcionalidad que también incorporaré. Además, puesto que mis potenciales clientes pueden no conocer muy bien el funcionamiento de las redes sociales, se mostrarán ayudas y sugerencias, de manera que puedan ayudar a que la publicación tenga mayor visibilidad.

6. Objetivos

El **objetivo principal** de este proyecto de fin de grado es el crear una herramienta que permita la gestión de varias redes sociales para la publicación o difusión de contenidos y monitorizar su impacto. Las características principales que debe tener son:

- Publicar un mismo contenido en las redes sociales que seleccione el usuario.
- Diseño llamativo e intuitivo con ayudas y/o sugerencias para el usuario en caso de detectar alguna incompatibilidad con alguna red social seleccionada.
- Visualización de las reacciones que ha tenido una publicación por parte de los usuarios de cada red social.

Para poder cumplir con estos objetivos, debemos tener en cuenta los **objetivos secundarios**:

- Aprender a utilizar las tecnologías que harán realidad esta aplicación.
- Desarrollar una base de datos MySQL junto con un API rest para las peticiones cliente-servidor.
- Conocer el funcionamiento de las distintas API de las redes sociales a utilizar y sus limitaciones.
- Aprender el funcionamiento del protocolo Oauth2 para que un usuario pueda iniciar sesión en esta plataforma web sin necesidad de registrarse, simplemente utilizando sus cuentas de otras redes sociales.
- Analizar el flujo de trabajo de las redes sociales con las que va a trabajar esta plataforma web.
- Proponer un flujo de trabajo ágil.
- Proponer una interfaz usable, accesible e intuitiva, pensada sobre todo para aquellas personas no profesionales ni acostumbradas al mundo del social media.
- Proporcionar un acceso sencillo, directo y rápido a las diferentes opciones que ofrece esta aplicación.
- Mostrar únicamente el contenido relevante para realizar las tareas, de manera que el usuario no se pierda por la aplicación y pueda así realizar las tareas en el menor tiempo posible.
- Desplegar la herramienta en un servidor abierto a internet, de manera que cualquier persona en cualquier parte del mundo pueda utilizarlo.

- Validar y comprobar que efectivamente el tiempo invertido en publicar contenido en varias redes sociales es menor utilizando esta aplicación que haciéndolo en cada red social de una en una directamente.

7. Metodología

Después de analizar distintas metodologías, he decidido que haré uso de la metodología SCRUM. Creo que esta es la metodología más adecuada para este proyecto. Con SCRUM, se realizan entregas parciales cada cierto periodo de tiempo, cada una de ellas con distintos avances dónde se va observando poco a poco cómo será el resultado final.

No es la primera vez que me enfrento a este tipo de metodología. Durante los años de carrera la he utilizado en distintos proyectos y siempre con resultados satisfactorios. Además, el hecho de estar ya familiarizado con ella puede ayudar a realizar este proyecto con éxito.

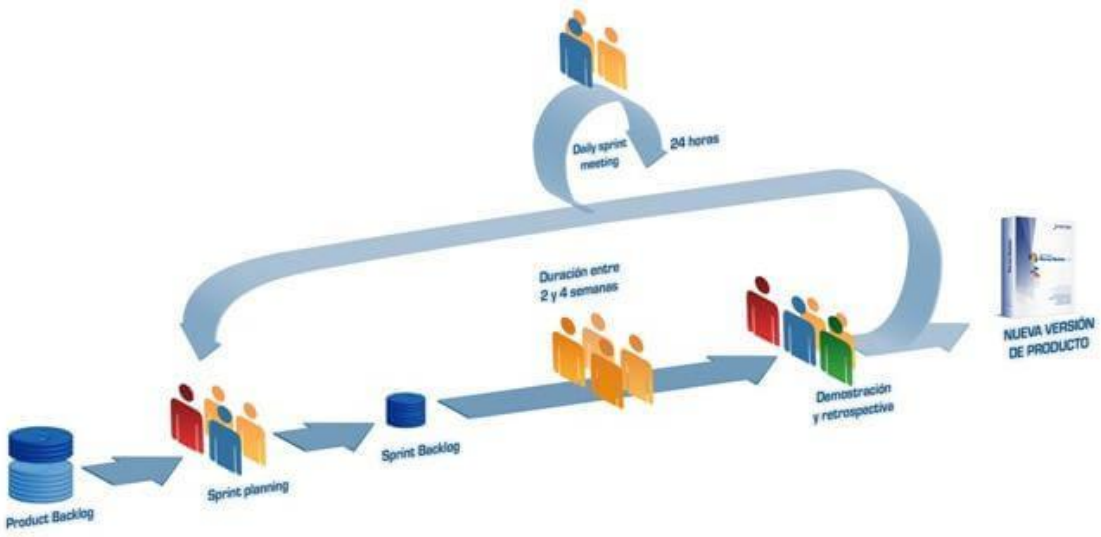


Figura 3. Metodología SCRUM
(https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html)

Con esta metodología, todos los entregables serán partes funcionales de lo que será el proyecto final. A medida que se van entregando, se revisan y se comprueba su correcto funcionamiento por parte del cliente.

Además, las diferentes tareas se ordenan teniendo en cuenta su prioridad. En cada sprint, se comprueba la prioridad de las tareas pendientes.

Para poder llevar a cabo el seguimiento tanto de las tareas realizadas, como de las que se están llevando a cabo y las que quedan por realizar, haré uso de la aplicación Trello.

Las primeras tareas se centrarán en el desarrollo del backend, tanto las relacionadas con el API rest como las diferentes pruebas, para poder comprobar así el correcto funcionamiento de las distintas APIs de cada red social. Una vez que la parte del backend esté casi terminada (siempre pueden surgir cambios, tanto a la hora de desarrollar el fontend como cambios en las políticas de uso y privacidad de las APIs externas), me centraré en ese momento, en el desarrollo frontend.

8. Análisis y especificación

8.1. Descripción general

En este apartado se definirán los requisitos y funcionalidades del proyecto. Para ello, haré uso del estándar IEEE 830. Este estándar comprende todas las tareas relacionadas con las necesidades y las condiciones a satisfacer en un software, de manera que podamos asegurar que se abordarán y se analizarán todas las áreas del proyecto.

Para ello, el estándar IEEE 830, divide los requisitos en “funcionales”, “no funcionales” y “otros requisitos”.

Los requisitos serán identificados mediante un código alfanumérico que tendrá la siguiente estructura TIPO-NÚMERO, donde cada elemento está descrito de la siguiente manera:

- Identificador
- Usuario
- Nombre
- Tipo (obligatorio u opcional).

Tras ello, se muestra la descripción detallada del requisito.

En cuanto a los distintos usuarios, la plataforma distingue dos tipos distintos:

- Cliente
- Administrador

8.2. Requisitos funcionales

En este subapartado, se van a definir los requisitos funcionales, agrupados cada uno de ellos en una tabla:

Identificador	RF-1
Usuario	Cliente y administrador
Nombre	Identificación
Tipo	Obligatorio
Habilitar el mecanismo por el cual un usuario que ya está registrado en el sistema puede acceder al mismo.	

Identificador	RF-2
Usuario	Cliente
Nombre	Vincular redes sociales
Tipo	Obligatorio
El cliente podrá vincular y desvincular sus redes sociales siempre que estas estén disponibles en la plataforma.	

Identificador	RF-3
Usuario	Cliente
Nombre	Publicar
Tipo	Obligatorio
El cliente podrá publicar texto o imágenes a las redes sociales que seleccione siempre que cumpla los requisitos de cada una de ellas (por ejemplo, en Instagram es obligatorio publicar una foto o vídeo en la publicación).	

Identificador	RF-4
Usuario	Cliente
Nombre	Programar publicación
Tipo	Obligatorio
El cliente podrá programar una publicación para un día y una hora concreta.	

Identificador	RF-5
Usuario	Cliente
Nombre	Eliminar publicación
Tipo	Obligatorio
El cliente podrá eliminar una publicación, bien de las ya realizadas o de las que aún están programadas (y nunca se llegaría a programar).	

Identificador	RF-6
Usuario	Cliente
Nombre	Editar publicaciones programadas
Tipo	Obligatorio
El cliente podrá editar las publicaciones que están programadas y todavía no se han publicado. Puede editar tanto el contenido de la misma, como la fecha y hora a la que está programada.	

Identificador	RF-7
Usuario	Cliente
Nombre	Consultar publicaciones.
Tipo	Opcional
El cliente tendrá la posibilidad de consultar todas sus publicaciones ya realizadas y visualizar desde un mismo sitio las reacciones que ha tenido en cada red social	

Identificador	RF-8
Usuario	Cliente
Nombre	Crear anotaciones
Tipo	Opcional
El cliente tendrá la posibilidad de crear anotaciones.	

Identificador	RF-9
Usuario	Cliente
Nombre	Editar anotaciones
Tipo	Opcional
El cliente podrá editar las anotaciones creadas.	

Identificador	RF-10
Usuario	Cliente
Nombre	Eliminar anotaciones
Tipo	Opcional
El cliente podrá eliminar las anotaciones creadas.	

Identificador	RF-11
Usuario	Cliente
Nombre	Activar notificaciones por email
Tipo	Opcional
El cliente podrá activar notificaciones en las notas creadas. Esta notificación o aviso, se le mandará como email.	

Identificador	RF-12
Usuario	Cliente
Nombre	Modificar datos
Tipo	Obligatorio
El cliente podrá modificar su nombre y el email al que le llegan las notificacioens.	

Identificador	RF-13
Usuario	Cliente
Nombre	Cerrar sesión
Tipo	Obligatorio
El cliente podrá cerrar la sesión en el dispositivo en el que se encuentra identificado	

Identificador	RF-14
Usuario	Cliente
Nombre	Eliminar cuenta
Tipo	Obligatorio
El cliente tendrá la posibilidad de dar de baja su cuenta de la plataforma web de Capsule.	

Identificador	RF-15
Usuario	Administrador
Nombre	Visualizar estadísticas
Tipo	Opcional
Desde el panel de administración se podrán consultar las estadísticas de las métricas clave definidas en el estudio de viabilidad.	

Identificador	RF-16
Usuario	Administrador
Nombre	Envío de emails
Tipo	Opcional
El administrador podrá enviar emails a los usuarios registrados en la plataforma.	

8.3. Requisitos no funcionales

Identificador	RNF-1
Nombre	Rendimiento
Las peticiones que realice un usuario deben tener un tiempo de respuesta menor a 3 segundos	

Identificador	RNF-2
Nombre	Seguridad peticiones cliente-servidor
El sistema garantizará una comunicación entre el cliente y servidor segura HTTPS (instalación de certificados SSL/TLS).	

Identificador	RNF-2
Nombre	Seguridad acceso al servidor
El sistema no será accesible a ninguna persona no autorizada.	

Identificador	RNF-3
Nombre	Envío de emails
El sistema será capaz de enviar emails a los usuarios.	

Identificador	RNF-4
Nombre	Copias de seguridad
El sistema debe realizar backups, tanto de la base de datos como de otras configuraciones, para que, en caso de ataque o pérdida de datos, pueda recuperarlos con el menor daño posible.	

Identificador	RNF-5
Nombre	Disponibilidad
El sistema debe estar disponible con un porcentaje cercano al 99%.	

Identificador	RNF-7
Nombre	Escalabilidad
El sistema debe ser escalable, permitiendo la incorporación de nuevas funciones sin afectar al rendimiento ni a la calidad.	

Identificador	RNF-8
Nombre	Estilo
Diseñar un estilo corporativo para el sistema. Las interfaces deben ser coherentes en todos los apartados del sistema.	

Identificador	RNF-9
Nombre	Usabilidad
Diseñar las interfaces pensando en ofrecer servicios intuitivos, que requieran el mínimo de intervenciones del usuario, con interfaces homogéneas y que generen una sensación grata en el uso y un buen recuerdo.	

9. Diseño

En este apartado, se incluye tanto el diseño de la arquitectura del servidor, su base de datos y el API Rest, como de las interfaces y la manera de interactuar con ellas.

9.1. Diseño de la persistencia

Para el almacenamiento de los datos, se utilizará una base de datos relacional SQL. El porqué de utilizar una base de datos de este tipo es por el hecho de que todos los datos van a ser consistentes. Todo lo que se va a almacenar, será igual para todos los usuarios. Además, es un lenguaje que he utilizado y con el que estoy acostumbrado.

Para gestionar las bases de datos, se hará uso de MariaDB, un sistema derivado de MySQL pero con licencia GPL (General Public License).

En la siguiente figura, se puede observar el diseño de la base de datos. Todos estos datos, serán devueltos a través de un API Rest.

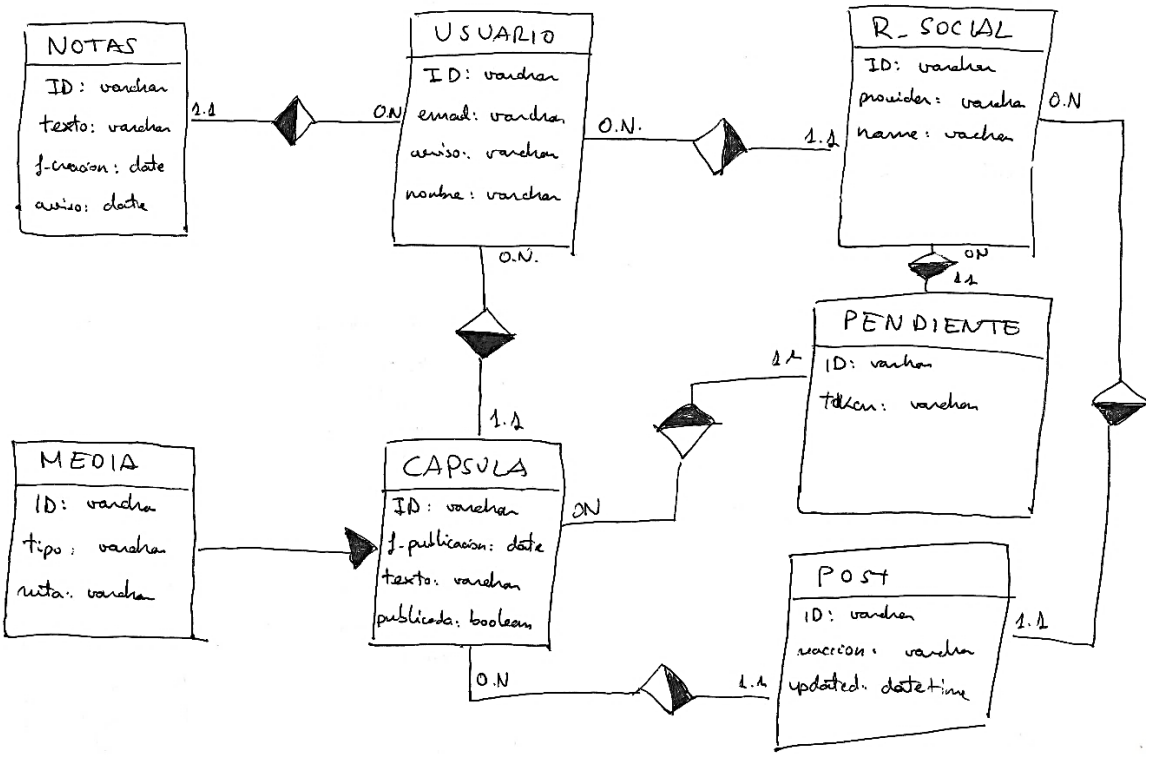


Figura 4. Diseño modelo Entidad Relación

9.2. Diseño arquitectura conceptual

En la siguiente figura se muestra el diseño de la arquitectura.

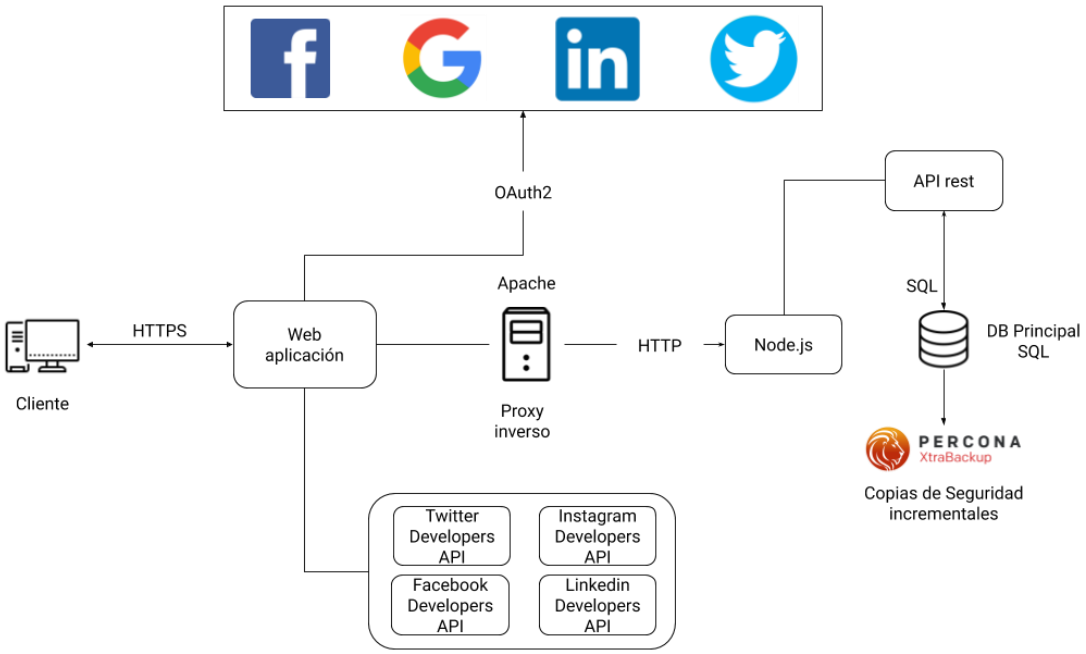


Figura 5. Diseño arquitectura backend

Los elementos principales que encontramos en la aplicación son:

- El **Cliente**, que, gracias al servicio Apache, se le servirá el contenido de la aplicación web.
- La propia **aplicación web**, que se comunicará tanto con el cliente como con el API Rest.
- El **API Rest**, mediante el uso de Node js.
- La **Base de datos SQL**, que se servirá a través del API Rest.
- **Percona XtraBackup**: para la realización de copias de seguridad incrementales.

Además, en el esquema, también queda constancia de las API externas con las que se conectará la aplicación, así como el uso del protocolo OAuth2 y las copias de seguridad.

9.3. Diseño API Rest

En este apartado se muestra el listado con el routing del API Rest, la definición de los parámetros y los datos/formatos que devuelve nuestra API.

9.3.1. Usuario

Ruta	/rest/user/:id
Método	GET
Parámetros	Id del usuario
Descripción	Obtiene un usuario concreto.

Ruta	/rest/users
Método	GET
Descripción	Obtiene un listado con todos los usuarios. Los resultados estarán paginados de 10 en 10.

Ruta	/rest/user
Método	POST
Cuerpo mensaje	Email: string (email del usuario). Nombre: string.
Descripción	Crea un nuevo usuario.

Ruta	/rest/user/:id
Método	PUT
Parámetros	Id del usuario
Cuerpo mensaje	Email: string (email del usuario). Nombre: string.
Descripción	Modifica los datos del usuario.

Ruta	/rest/user/:id
Método	DELETE
Parámetros	Id del usuario
Descripción	Elimina el usuario

9.3.2. Capsula

Una capsula es la manera de llamar al elemento que agrupa una misma publicación en distintas redes sociales.

Ruta	/rest/capsule/:id
Método	GET
Parámetros	Id de la capsula
Descripción	Obtiene una capsula concreta.

Ruta	/rest/capsule/byUser/:id
Método	GET

Parámetros	Id del usuario
Descripción	Obtiene todas las capsulas de un usuario, paginadas de 10 en 10.

Ruta	/rest/capsule
Método	POST
Cuerpo mensaje	F_publicacion: fecha en la que se publicará Texto: texto de la capsula
Descripción	Crea una nueva capsula.

Ruta	/rest/capsule/:id
Método	PUT
Parámetros	Id de la capsula
Cuerpo mensaje	F_publicacion: fecha en la que se publicará Texto: texto de la capsula
Descripción	Modifica la fecha de una capsula.

Ruta	/rest/capsule/:id
Método	DELETE
Parámetros	Id de la capsula
Descripción	Elimina el usuario

9.3.3. Media

Aquí se almacenarán temporalmente las imágenes o vídeos que un usuario quiera publicar en las redes sociales. Una vez realizada la publicación, como dichas imágenes ya estarán en sus respectivas redes sociales, se eliminarán.

Ruta	/rest/media/byCapsule/:id
Método	GET
Parámetros	Id de la capsula
Descripción	Obtiene todas las imágenes de una capsula.

Ruta	/rest/media
Método	POST
Cuerpo mensaje	Tipo: para diferenciar si se trata de un vídeo o una foto Ruta: donde está almacenado el contenido en el servidor, para luego obtener de ahí el archivo.
Descripción	Guarda los datos tras la subida de una foto.

Ruta	/rest/media/:id
Método	DELETE

Parámetros	Id de una imagen
Descripción	Elimina una imagen

9.3.4. Post

Aquí se almacenará la ID que asignará cada red social a un post. Una vez publicado desde capsule, Facebook, por ejemplo, asigna una id al nuevo post creado. En esta tabla la almacenaré, para así luego, poder obtener datos sobre ella.

Ruta	/rest/post/byCapsule/:id
Método	GET
Parámetros	Id de la capsula
Descripción	Obtiene todos los posts de una capsula.

Ruta	/rest/post
Método	POST
Cuerpo mensaje	ID: id devuelta por la red social
Descripción	Guarda la id donde la red social ha subido la publicación.

9.3.5. Red Social

Ruta	/rest/r_social/byUser/:id
Método	GET
Parámetros	Id del usuario
Descripción	Obtiene todas las redes sociales de un usuario.

Ruta	/rest/r_social
Método	POST
Cuerpo mensaje	ID: id devuelta por la red social tras devolver los datos al asociarla con esta plataforma web. Provider: qué tipo de red social se trata (“Facebook”, “Instagram”…).
Descripción	Guarda la información de la red social que el usuario ha asociado.

Ruta	/rest/r_social/:id
Método	DELETE
Parámetros	Id de la red social
Descripción	Elimina (desvincula) esa red social del usuario que la añadió.

9.3.6. Notas

Ruta	/rest/notas/byUser/:id
Método	GET

Parámetros	Id del usuario
Descripción	Obtiene todas las notas de un usuario.

Ruta	/rest/notas /:id
Método	GET
Parámetros	Id de la nota
Descripción	Obtiene una nota concreta.

Ruta	/rest/notas
Método	POST
Cuerpo mensaje	Texto: el texto de la nota Aviso: por defecto a null. Si el usuario activa el aviso, se guardará la fecha en la que quiere ser avisado.
Descripción	Crea una nueva nota.

Ruta	/rest/notas/:id
Método	PUT
Parámetros	Id de la nota
Cuerpo mensaje	Texto: el texto de la nota Aviso: por defecto a null. Si el usuario activa el aviso, se guardará la fecha en la que quiere ser avisado.
Descripción	Modifica una nota.

Ruta	/rest/notas /:id
Método	DELETE
Parámetros	Id de la nota
Descripción	Elimina una nota.

9.3.7. Pendientes

Ruta	/rest/pendientes /
Método	POST
Cuerpo mensaje	Id de la capsula, de la red social y el token
Descripción	Añade una publicación a las pendientes.

Ruta	/rest/pendientes /byUser/:id
Método	GET
Parámetros	Id del usuario
Descripción	Devuelve los datos de las publicaciones pendientes paginados de 10 en 10.

Ruta	/rest/pendientes /:id
Método	GET
Parámetros	ID
Descripción	Devuelve los datos de una publicación pendiente.

Ruta	/rest/pendientes /:id
Método	DELETE
Parámetros	ID
Descripción	Elimina una publicación pendiente.

9.3.8. Login

Ruta	/rest/login
Método	POST
Cuerpo mensaje	Email: email del usuario. Token: token de la red social. Estos datos serán devueltos por el protocolo oauth2, y se enviarán a esta petición POST, donde se comprobará el token y se identificará al usuario por su email.
Descripción	Crea una nueva nota.

9.4. Diseño arquitectura tecnológica frontend/backend

En este apartado se especifican las propuestas tecnológicas que serán utilizadas para el desarrollo de este proyecto. Diferenciamos entre frontend y backend.

9.4.1. Frontend

Se trata de lo que el cliente ve y con lo que interacciona. El frontend de Capsule estará formado por archivos HTML, CSS y JavaScript. Sin embargo, a la hora de desarrollarlo, se hará con Angular.

Angular es un framework de desarrollo web, de código abierto y mantenido por Google, que trabaja con el lenguaje TypeScript (desarrollado y mantenido por Microsoft), lenguaje que extiende la sintaxis de Javascript. Angular se utiliza para la creación de aplicaciones web de una sola página.

Una de las ventajas de utilizar Angular es el hecho de que te permite organizar todo el código a la hora de trabajar, por componentes. De esta manera, obtenemos un código bastante ordenado, cómodo y fácil para trabajar, además de una mejora en el rendimiento, ya que puede

cargar únicamente el componente o las partes del código que se le soliciten, sin tener que volver a cargar toda la página completa.

Además, a la hora de sacar el proyecto a producción, con un simple comando, se genera todo el proyecto en archivos HTML, CSS y JavaScript ya optimizados, para así, de una manera rápida y sencilla, poder llevarlos al servidor sin necesidad de modificar o adaptar nada más.

Para la parte más visual, se va a utilizar **Angular Material**, un framework CSS desarrollado por Google que ofrece todas las características de Material Design (normativa de diseño enfocada en el desarrollo de aplicaciones multidispositivo desarrollada por Google y utilizada principalmente en su sistema operativo Android). Angular Material trae consigo todas las ventajas del Material Design en forma de componentes para Angular, de manera que se puede integrar sin problemas a este proyecto.

Para algunos detalles donde Angular Material se queda un poco corto, se utilizará **MDBootstrap**, puesto que comparte la misma característica de "Material Design" y, por tanto, me puede servir como apoyo a Angular Material.

9.4.2. Backend

Es decir, respecto todo aquello que el cliente no ve, lo que hay por detrás y que hace que la aplicación funcione. Para el backend de Capsule, se utilizarán las siguientes tecnologías:

- **MariaDB** (MySQL): Para la base de datos, ya que se trata de una base de datos relacional. Es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL.
- **NodeJS**: entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor.
- **Express**: framework para NodeJS diseñado para la creación de aplicaciones web y APIs. En este caso, se utilizará para todas las peticiones del API Rest.
- **Percona XtraBackup**: para la gestión de copias de seguridad de toda la base de datos.
- **Apache**: para el servidor web, y desde donde se enviarán y se recibirán todas las peticiones del cliente, configurando para ello un proxy inverso con el API Rest.

En cuanto a los servicios de terceros, se hará uso del protocolo **OAuth2**, para todas las funciones donde se interaccionará con Redes Sociales externas; esto es: el **Login** y registro, la **subida de posts** a las redes sociales y el **envío de emails** por parte del administrador.

9.5. Diseño Interacción o Experiencia de Usuario

La experiencia de usuario es uno de los aspectos más importantes a la hora de desarrollar una aplicación, puesto que es la manera en la que esta entrará por los ojos de los clientes, es decir, cómo perciben y reaccionan y qué sensaciones les causa. La elección de una buena experiencia de usuario puede ser un factor de fracaso o éxito en el proyecto.

Uno de los principales objetivos será conocer el segmento de clientes.

Puesto que la mayoría de los potenciales clientes serán personas que seguramente ya habrán trabajado con redes sociales y estarán familiarizados con ellas, será necesario ofrecer una interfaz con características similares, como vemos en las figuras 6, 7 y 8; de manera que se consiga una plataforma web más intuitiva.

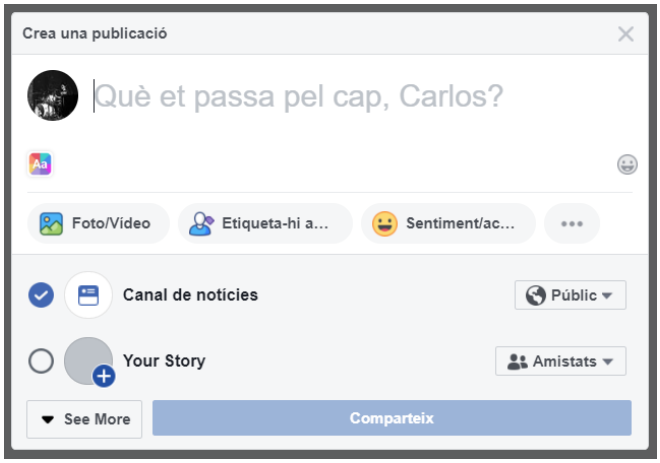


Figura 6. Facebook

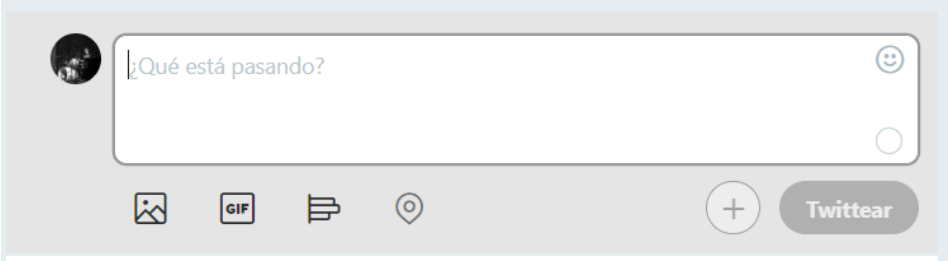


Figura 7. Twitter

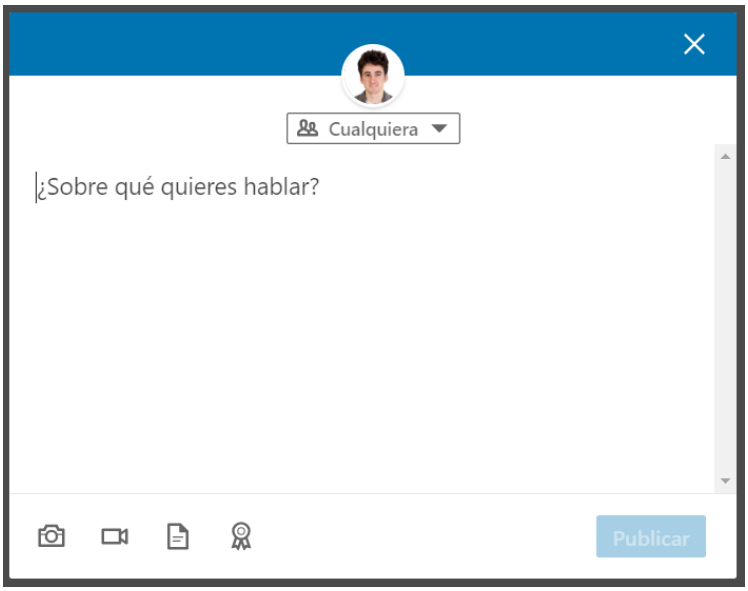


Figura 8. Linkedin

Como vemos en la versión de escritorio de las redes sociales Facebook, Twitter y Linkedin, el diseño a la hora de crear una publicación nueva es muy similar. En primer lugar, se escribe el texto, y después se seleccionan los archivos multimedia que quieres compartir junto con el texto. Por este motivo, el mecanismo para publicar una nueva publicación en Capsule, deberá ser similar.

Otro de los elementos que tiene mucha importancia dentro de la experiencia de usuario es el tiempo de carga, sobre todo en los apartados donde se pueden mostrar una gran cantidad de datos, como podría suceder cuando un usuario quisiese ver todas sus publicaciones realizadas. Pueden ser solo unas pocas, o pueden ser muchísimas, por este motivo, no se deben mostrar todas de golpe, puesto que, si fuera así, cuantas más publicaciones tuviera un usuario, más tiempo tardaría la plataforma en mostrarlas todas. Para ello, será mucho mejor que los resultados se carguen y devuelvan poco a poco. Además, para evitar hacer uso de un menú de paginación y que el usuario tenga que realizar más clicks, los elementos se irán cargando conforme se haga scroll vertical en la página. Este funcionamiento es conocido como “Lazy Loading” y, de esta manera, la sensación será que se ha cargado todo el contenido de una sola vez, aunque en realidad no se hace hasta el momento en el que el usuario hace uso de él.

9.6. Diseño Interfaces

El diseño de interfaces es la parte donde los RF y RNF se materializan, es decir, es la parte de la ingeniería donde se desarrollan los mecanismos físicos mediante los cuales los distintos usuarios podrán interaccionar con el sistema. Teniendo en cuenta siempre el segmento de clientes al que va dirigida la aplicación, no podemos olvidar que las interfaces deben ser fáciles (tanto de utilizar como de entender) y accesibles para todos.

Con estas dos ideas en mente, junto con los requisitos funcionales anteriormente descritos, se han diseñado los siguientes Wireframes:

WF-1: Login

Desde donde un usuario podrá acceder a Capsule. Cumple con el Requisito Funcional RF-1.

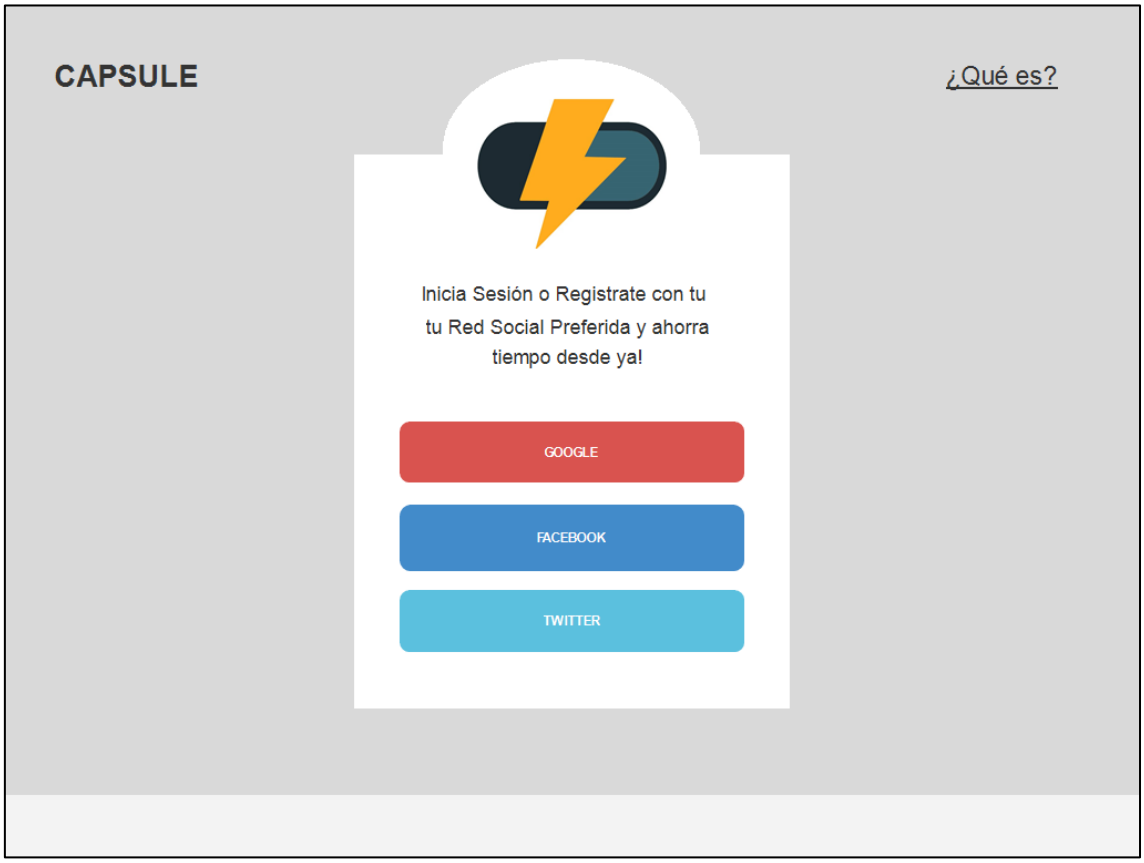


Figura 9. Wireframe WF-1. Login

WF-2: Datos del usuario

Desde esta sección el usuario podrá modificar sus datos y gestionar las redes sociales que tiene sincronizadas con la plataforma web. En este wireframe se encuentran los Requisitos Funcionales RF-2, RF-12 y RF-14.

También podemos observar el *header*, donde se encuentra el botón para cerrar sesión (Requisito Funcional RF-13).

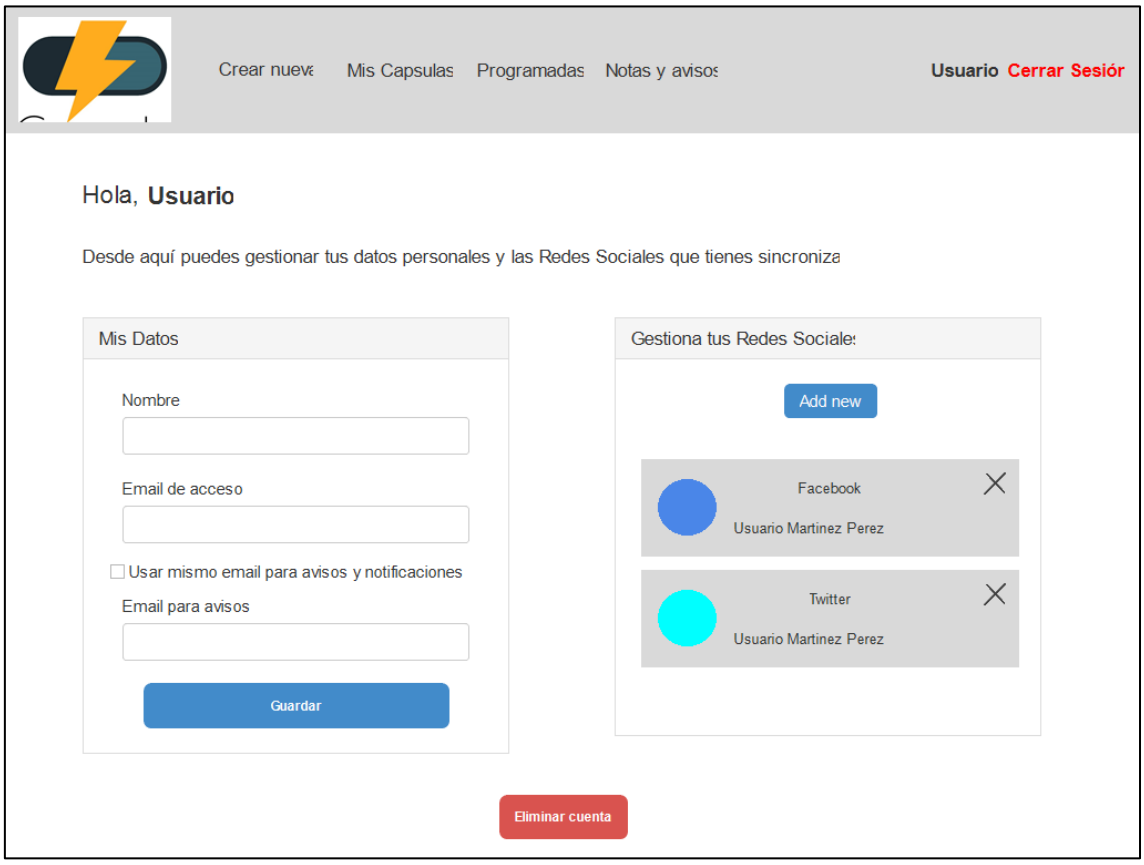


Figura 10. Wireframe WF-2. Datos del usuario.

WF-3: Crear nueva publicación

Cumpliendo el Requisito Funcional RF-3.

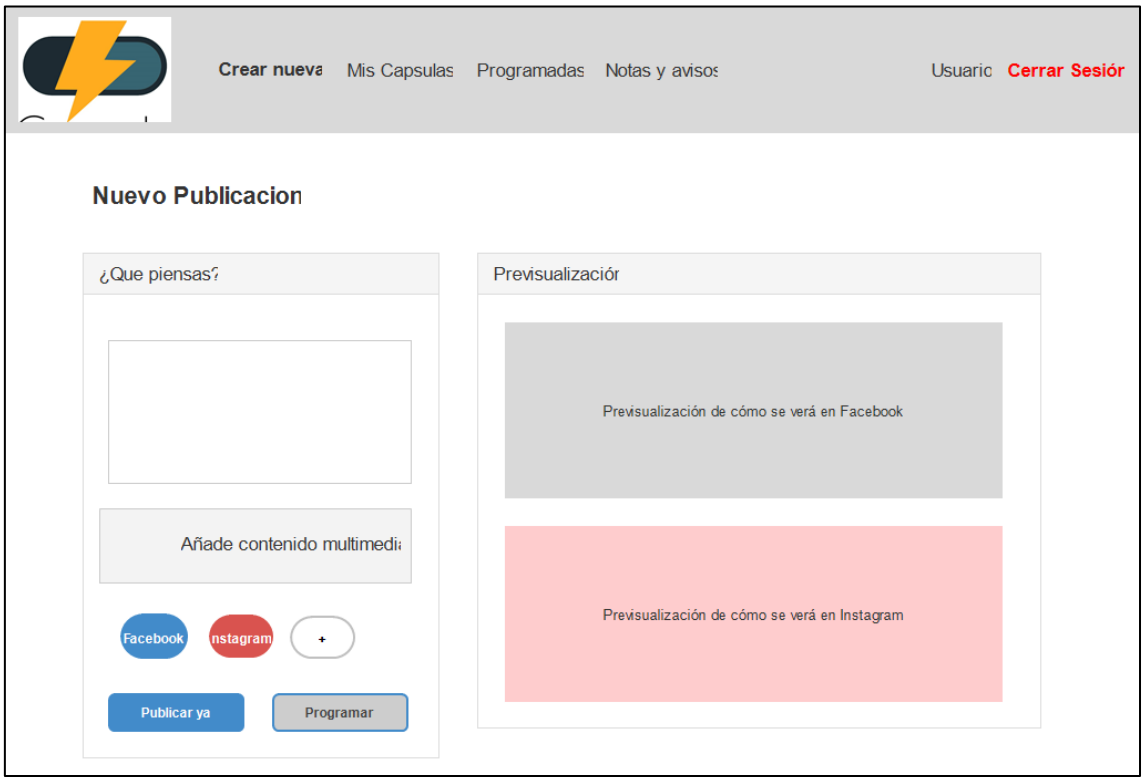


Figura 11. WF-3. Crear nueva publicación

WF-4: Programar publicación/aviso

Cumpliendo con el RF-4, desde este apartado se podrá seleccionar la fecha en la que se publicará el contenido. Este mismo "pop-up" se utilizará también para los avisos de las notas.

Figura 12. WF-4. Programar publicación/aviso

WF-5: Mis capsulas

En esta sección encontramos los requisitos funcionales RF-5 y RF-7, y es donde un usuario podrá consultar sus publicaciones realizadas desde Capsule y/o eliminarlas.

En el lateral derecho se encontrarán un campo de búsqueda para filtrar los resultados.

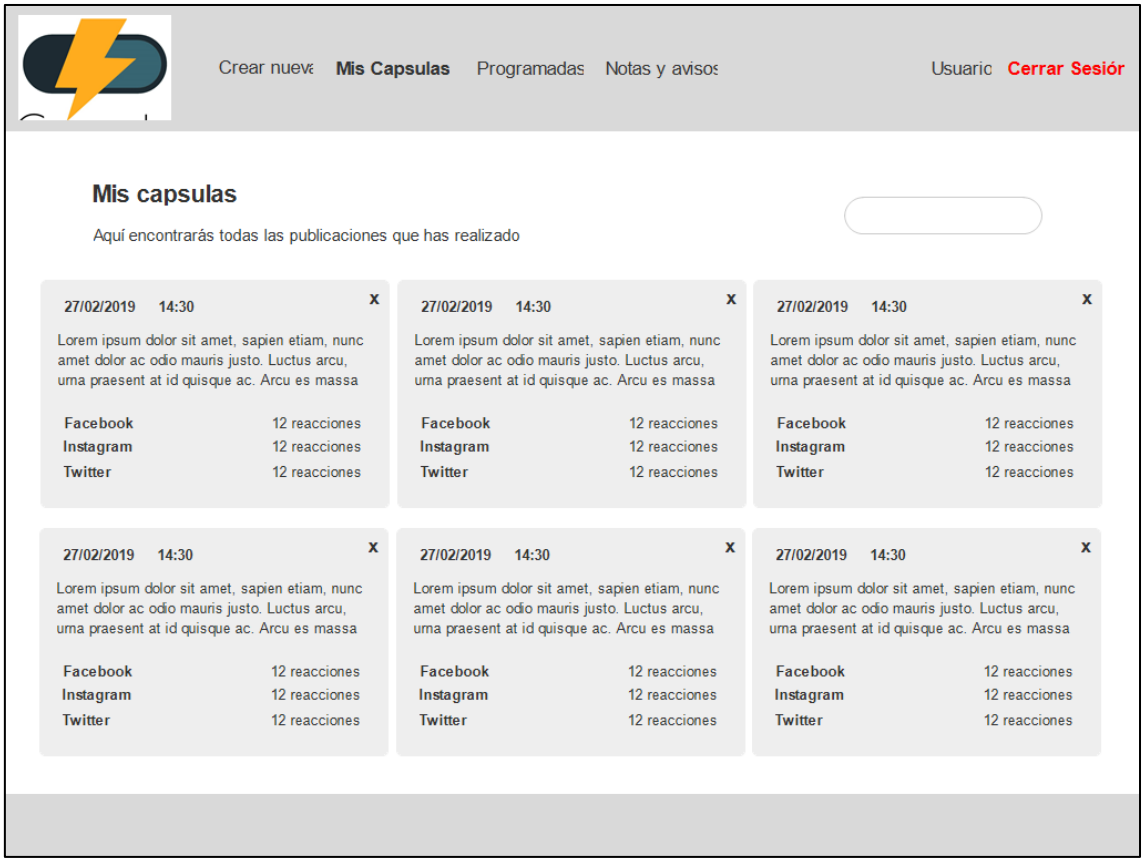


Figura 13. WF-5. Mis Capsulas.

WF-6: Capsulas programadas

Cumpliendo con el RF-6. Es el lugar donde encontrará todas sus futuras publicaciones.

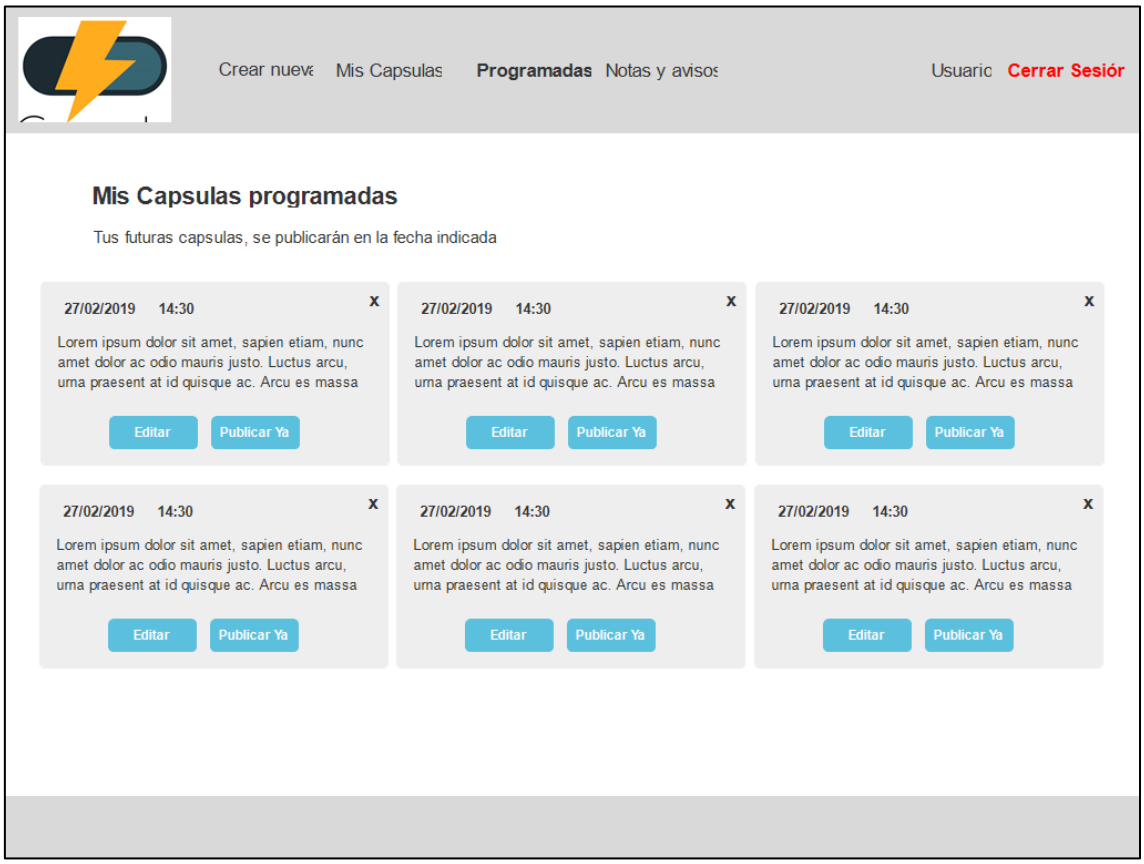


Figura 14. WF-6. Cosultar publicaciones programadas

WF-7: Mis anotaciones

En esta sección, un usuario podrá crear nuevas notas, editarlas, eliminarlas y activar los avisos. Con todo ello, cumplimos los Requisitos Funcionales RF-8, RF-9, RF-10 y RF-11

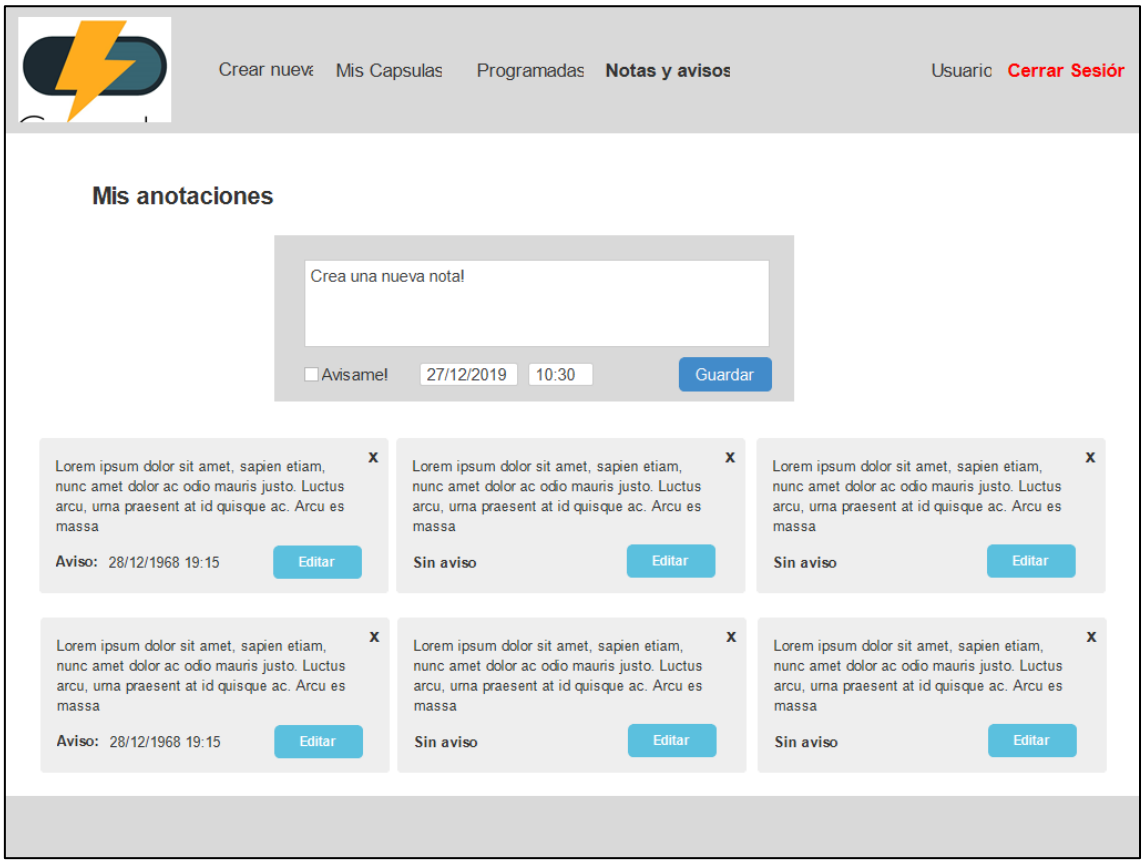


Figura 15. WF-7. Mis anotaciones

WF-8: Panel del administrador

Desde esta sección, el administrador puede comprobar las estadísticas de las métricas clave definidas y también puede enviar un email a todos los usuarios registrados en Capsule, a modo de informar sobre actualizaciones, novedades o problemas. Estos serían los requisitos funcionales RF-15 y RF-16

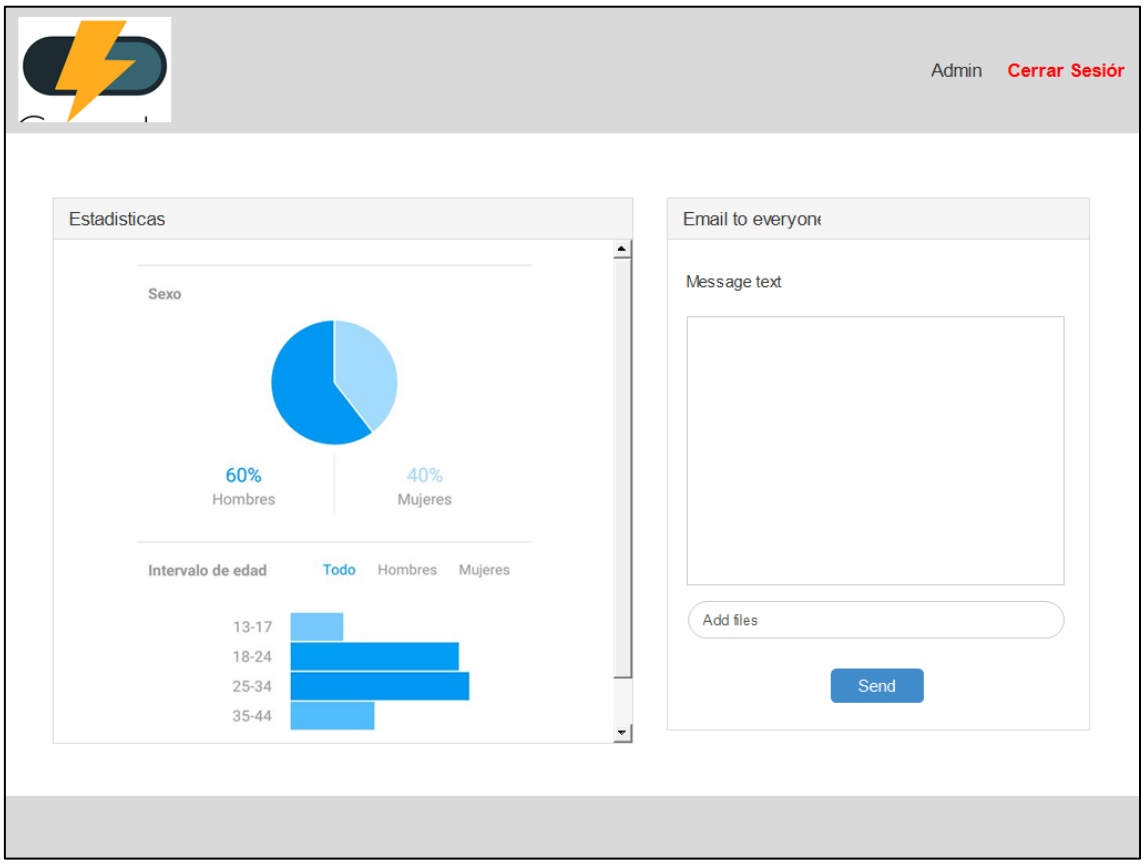


Figura 16. WF-8. Panel del administrador

9.7. Guías de estilos

Logotipo:

Se trata del icono de una capsula (puesto que la aplicación contiene o “encapsula” varias redes sociales en un mismo sitio) con el icono de un rayo o relámpago sobre él, para dejar claro que es algo rápido, algo que te ayudará a ganar tiempo a la hora de publicar en redes sociales.

Figura 17. Logotipo Capsule

Colores:

- **Principales**

#232f34 **#4a6572** **#f9aa33**

- **Otros colores**

#e0e0e0 **#424242** **#b00020**

Líneas *Texto* *Alertas*

Tipografía:

Se utilizará **Open Sans**, excepto para el logotipo, donde se utilizará **Poiret One**.

Ejemplo Poiret One	Ejemplo Open Sans
Penultimate The spirit is willing but the flesh is weak SCHADENFREUDE 3964 Elm Street and 1370 Rt. 21 The left hand does not know what the right hand is doing.	Penultimate The spirit is willing but the flesh is weak SCHADENFREUDE 3964 Elm Street and 1370 Rt. 21 The left hand does not know what the right hand is doing.

El resto de los elementos y del css en general, seguirán el uso y recomendación de **Angular Material.**

10. Implementación

Teniendo en cuenta que la metodología utilizada es **Scrum**, aquí voy a detallar los distintos **Sprints** que han tenido lugar a lo largo del desarrollo del proyecto.

10.1. Sprint 1: Entorno de desarrollo

Para el entorno de desarrollo, y trabajar cómodamente se ha utilizado un ordenador portátil con **Ubuntu 18.04**, 16GB de ram, y un procesador Intel i7-7700HQ.

Como editor de código, se va a utilizar **Atom**, un editor multiplataforma con integración con GitHub, junto con la extensión **emmet**, que facilita bastante el trabajo a la hora de escribir la parte en HTML.

Para el **Backend**, teniendo en cuenta una futura migración a un servidor real y abierto a internet, he decidido implementarlo en una **Raspberry PI model 3B**, puesto que por su limitado hardware y especificaciones, podré comprobar y conocer de mejor forma cómo se comportaría si, en un futuro, migrara toda la aplicación a un servidor abierto a internet (como por ejemplo, un VPS de OVH). Además, se van a utilizar dos Raspberry Pi model 3B: una como servidor de producción y otra como servidor de pruebas.

Figura 18. Raspberry pi de producción a la izquierda (la blanca) y otra de pruebas (negra) a la derecha. En el centro, el router.

Como ya se ha comentado, para el API Rest, se ha utilizado **Node**, junto con los siguientes modulos:

- **Express**: para poder utilizar el protocolo HTTP en las peticiones.
- **Body Parser:** para convertir las peticiones en objetos JSON.
- **Mysql:** para la conexión con la base de datos MariaDB.
- **JWT (JSON Web Tokens)**: para hacer uso de tokens en las peticiones y poder autenticar a los usuarios de la aplicación.

Para utilizar JWT, en el servidor se crea la función, en la que se incluyen los datos que se van a necesitar (en ese ejemplo, la id del usuario), y se devuelven en forma de token, junto con una clave y un tiempo de expiración.

Para poder acceder desde el frontend y obtener, en este caso, la id del usuario, será necesario decodificar el token haciendo uso de la función **decode().**

```
var jwt = require('jsonwebtoken');

exports.createToken = function(user){
        var payload = {
                "id":
        };
        return jwt.sign(payload,
};
```

Figura 19. Ejemplo de configuración de JSON Web Tokens.

10.2. Sprint 2: Implementación API Rest

Este segundo Sprint, se ha centrado en configurar todas las rutas principales, para así poder empezar más adelante, a trabajar de una forma “más real” el frontend. Más real, puesto que ya tendríamos las rutas para el login, crear usuarios o crear nuevos posts (en la base de datos) entre otros.

Para ello, se ha configurado una base de datos MariaDB con las tablas necesarias. Una vez creadas, se ha instalado Apache en la Raspberry y PHPMyAdmin.

Apache, para poder utilizar un proxy inverso y acceder a las peticiones del API Rest (y para en un futuro, poder disponer también de la aplicación web), y PHPMyAdmin para poder realizar, comprobar y visualizar de una forma fácil y rápida, desde cualquier navegador los datos de la

base de datos, y la correcta inserción, actualización o borrado de las peticiones que hago del frontend a través del API Rest.

Una vez configurado Apache y PHPMyAdmin, en el servidor Node se han creado todas las peticiones (GET, POST, PUT y DELETE) de todas las tablas con las rutas y parámetros necesarios.

```
login: function (req, res){
        let datos = req.body;
        if(!datos){
                return res.status(400).send({error:true, message: 'Please provide correct data'});
        }
        else{
                sql.query('SELECT * FROM usuario WHERE email = "'+datos.email+'"', function (error, results
                        if(results[0]!=null){
                                return res.status(200).send({token: service.createToken(results[0].ID)});
                        }
                        else{
```

Figura 20. Ejemplo de función (login), que conecta con la base de datos y devuelve un token.

En el archivo de configuración del servidor, se encuentran todas las rutas y los métodos:

```
//USUARIOS
var usuariosFile = require('./usuarios');

app.get('/users', usuariosFile.getUsers);
app.get('/user/:id', usuariosFile.getUser);
app.post('/user', usuariosFile.newUser);
app.put('/user/:id', usuariosFile.updateUser);
app.delete('/user/:id', usuariosFile.deleteUser);

//NOTAS
var notasFile = require('./notas');

app.get('/notas/byUser/:id', notasFile.byUser);
app.get('/notas/:id', notasFile.getNota);
app.post('/notas', notasFile.newNota);
app.put('/notas/:id', notasFile.updateNota);
app.delete('/notas/:id', notasFile.deleteNota);
```

Figura 21. Ejemplo de algunas rutas, que llaman a la función que se encuentra en distintos archivos del API Rest.

10.3. Sprint 3: Login

Una de las principales tareas en cualquier aplicación web es que el login funcione, puesto que, sin él, sería imposible acceder a la plataforma.

Las rutas ya están creadas del anterior Sprint, sin embargo, como se ha optado por dejar de lado el login “tradicional” de usuario y contraseña, y hacerlo únicamente con redes sociales y

servicios de internet a través del protocolo OAuth2, se ha tenido que hacer uso de las siguientes librarías:

- **Angularx-social-login**: permite la autenticación de usuarios a través de Google, Facebook y Linkedin.
- **Firebase:** plataforma web de Google, que, entre otras cosas, permite configurar OAuth2 para autenticarte con algunas plataformas web. En este caso, se utilizará sólo para Twitter.

El porqué de utilizar estas dos es porque la primera, angularx-social-login, consiste en una libraría que, después de instalarla e incluirla en el proyecto a través de npm, dispones de toda la funcionalidad de OAuth2 en el mismo. Es una librería que simplemente aporta las llamadas y peticiones necesarias a cada red social para poder obtener los datos de los usuarios. Sin embargo, esta no incluye la red social Twitter.

La segunda, Firebase, sí permite la autenticación a través de Twitter (y otras redes sociales, entre las que se encuentran también Facebook). Aun así, se ha preferido utilizar Firebase únicamente para Twitter, y mantener el resto de las redes sociales con la anterior, puesto que con ella se realizan todas las peticiones directamente desde la plataforma, sin depender de otra de terceros.

A pesar de esto, para poder obtener los datos de inicio de sesión de los usuarios, ha sido necesario crear en cada red social una cuenta de desarrollador, y configurar los datos que cada una pedía. Tras esto, se obtiene un ID de cliente y una clave secreta, que es la que se utilizará en las peticiones OAuth2.

Los datos que se reciben después de iniciar sesión en una red social a través de OAuth2 se muestras de la siguiente manera:

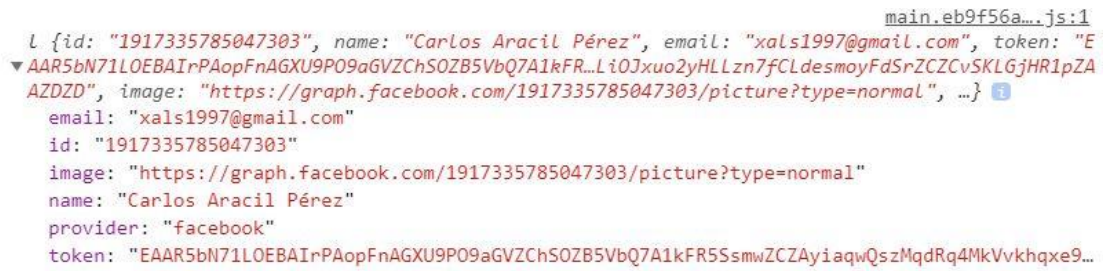


Figura 22. Ejemplo de respuesta OAuth2 con Facebook.

En el caso de Facebook, se recibe:

- Email
- ID
- Imagen de perfil
- Nombre
- Token

El token se utilizará para hacer peticiones con la red social. Cada red social devuelve unos datos, algunas como Twitter, utilizan dos tokens, uno de acceso y otro secreto.

Una vez obtenidos estos datos, si el usuario es nuevo, se guarda en la base de datos. Se comprueba el email, y se guarda en la tabla usuarios campos como su nombre y su email y se genera de forma automática y aleatoria una ID. Con esta ID y con un token propio de la aplicación Capsule (generado por JWT, comentado anteriormente), se identifica el usuario en la plataforma y éste ya podrá hacer uso de ella. El token que obtiene de la red social en cuestión se utiliza para hacer peticiones contra dicha red, mientras que el token de Capsule, se utiliza para poder acceder a la plataforma y hacer uso de sus funciones.

10.4. Sprint 4: Publicar a través de una petición OAuth2

En este sprint, simplemente se quiere comprobar que es posible realizar una publicación desde "fuera" de la red social. Para ello se va a optar por hacerlo a través de Twitter, pues es la que menos trabas pone.

Antes de nada, es necesario disponer, a parte del ID de la aplicación y de la clave secreta (ya utilizadas para el login), de permisos de escritura. Por defecto, las redes sociales sólo conceden permisos de lectura (suficientes para iniciar sesión, pero no para poder publicar en nombre de un usuario).

Cada red social pide unos permisos distintos, y exige también unos requisitos específicos para poder obtener este permiso.

En el caso de Facebook y Instagram, piden un vídeo de cómo funciona la aplicación, el motivo del porqué, dejar claro qué se va a hacer, cómo y de qué manera, disponer de una política de privacidad y de condiciones, y finalmente tener la aplicación web disponible en internet para que ellos puedan comprobar su funcionamiento.

Por tanto, se ha decidido dejarlas para más adelante (cuando se tenga algo más desarrollado que pueda poner accesible en internet) y centrarse en Twitter, ya que esta última, simplemente exige un correo explicativo del porqué necesitas permisos de escritura.

En este caso, después de escribir los motivos del porqué, comentando que se trataba de un proyecto universitario, Twitter concedió los permisos.

10.4.1. Texto

Una vez con los permisos necesarios, se siguió la documentación que Twitter dispone en su página para desarrolladores [14] para publicar una entrada de texto.

Para comprobar el funcionamiento, se utilizó la aplicación Postman. Para realizar un post en Twitter necesitamos:

- La **consumer_key** de nuestra aplicación
- Una clave generada automáticamente llamada **nonce**, que es aleatoria. Se puede generar desde la propia aplicación, y sirve para evitar realizar muchas peticiones de golpe, ya que cada petición necesita una clave aleatoria generada.
- El **token de acceso** del usuario, obtenido una vez este ha hecho login.
- El **código privado** de nuestra aplicación.
- El **secreto** del usuario.

El código privado de nuestra aplicación y el token secreto del usuario, son dos códigos que no deben ser compartidos ni enviados en la petición. El código privado de la aplicación se obtiene en la consola de desarrollador de twitter, junto con la clave pública o **consumer key** de la aplicación.

El secreto del usuario se devuelve en la petición oauth, en la misma que recogemos el token de acceso.

Con estos dos datos, es necesario crear una variable llamada **oauth_signature**. Primero se crea una “firma” base, creada por la url, la consumer key y el token de acceso del usuario. Se concatenan con “&” y se codifican los caracteres en formato de porcentajes (en este formato, símbolos como el “=” son “%3D” y el espacio, “%20“) .

Una vez tenemos esto, se hace lo mismo con el código privado de la aplicación y el secreto del usuario. Luego ambas cadenas de texto se concatenan y se codifican en base 64.

Una vez realizado, se aplica una función hash HMAC-SHA1 y se obtiene la que es la "firma", para autenticar la petición.

Todo esto está explicado y se puede encontrar la manera de obtener la clave en: https://developer.twitter.com/en/docs/basics/authentication/guides/creating-a-signature.html

Sin embargo, Postman incluye ya una opción para obtener esta clave sin necesidad de realizar todas las operaciones anteriores.

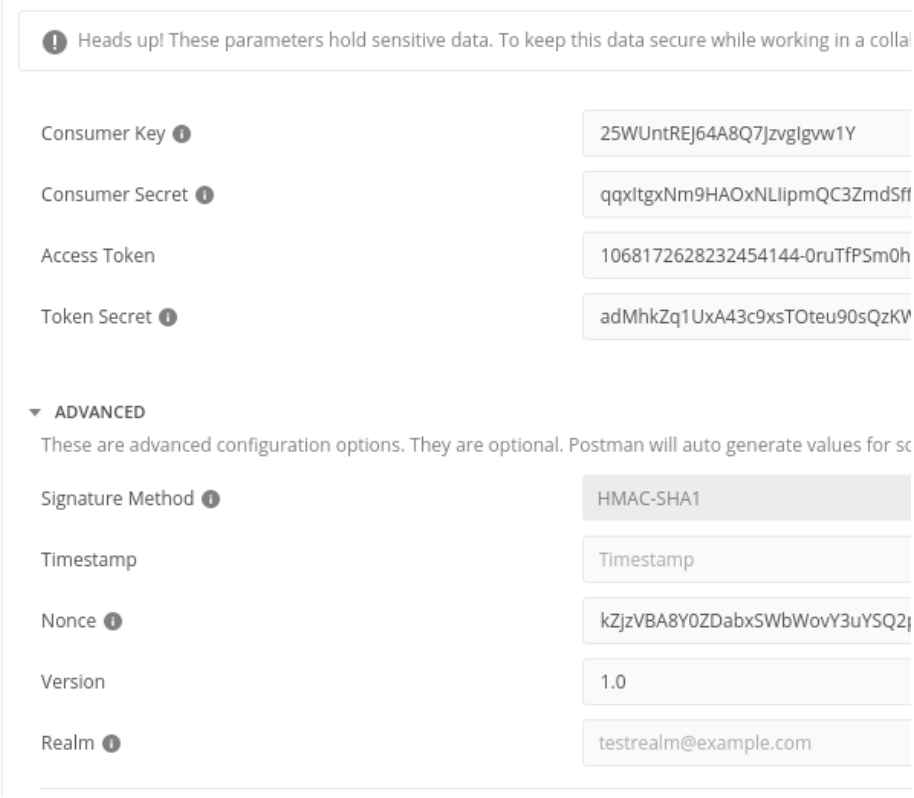


*Figura 23. Creación de la **oauth_signature** a través de Postman.*

Con los siguientes datos, podemos realizar una petición POST, añadiendo el parámetro **?status=** junto con el texto a publicar en la URL. En este caso, se va a escribir ***"A tope con el TFG"***, en la cuenta de Twitter **@xals1997**. La "firma" obtenida anteriormente, Postman la incluye de forma automática en la cabecera "Authorization".

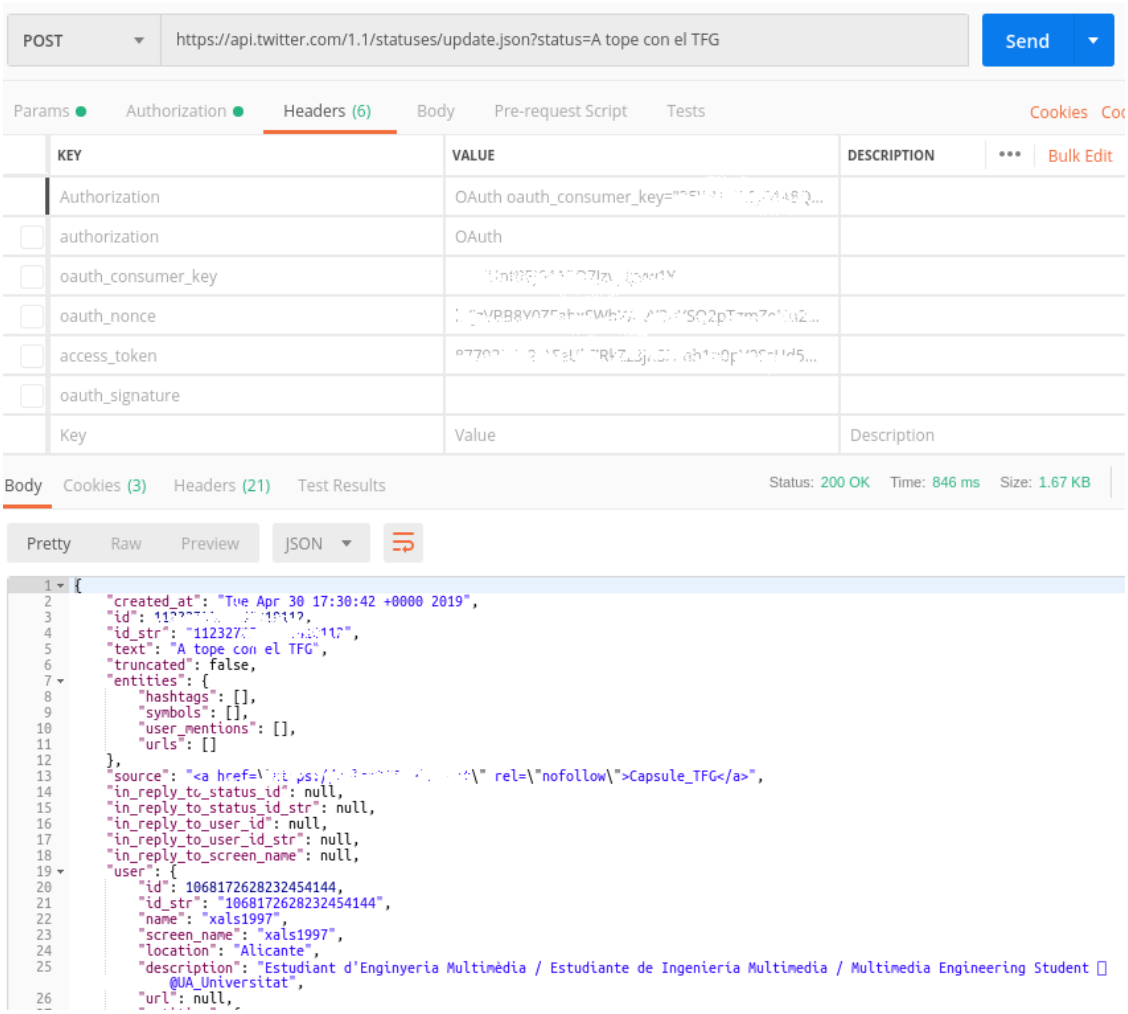


Figura 24. Ejemplo para publicar en Twitter a través de OAuth2 con la aplicación Postman.

Twitter nos devuelve los datos de Tweet, y comprobamos que, efectivamente, éste se ha publicado:

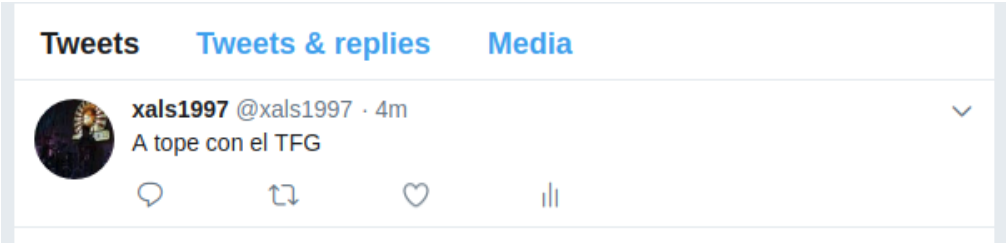


Figura 25. Tweet publicado a través de Postman.

10.4.2. Texto con imágenes

Para publicar el tweet con imágenes, primero hay que realizar una petición POST con la imagen y sus datos [15]. Para ello, se necesita, además de los datos y autenticaciones de OAuth2 utilizadas anteriormente:

- **Media:** el archivo.
- **Media_data:** el archivo codificado en base64.

El archivo codificado en base64 se puede obtener a través del input de tipo file de un formulario, entre otras maneras, y se envía como string en la petición. Como veis, es bastante extenso:

Figura 26. Obtención de datos de un archivo a través de la consola del navegador Chrome.

Con estos datos, realizamos la petición POST, y en la respuesta devuelta por el API de Twitter, encontraremos la ID, que es temporal y expira en los segundos indicados, de la imagen:

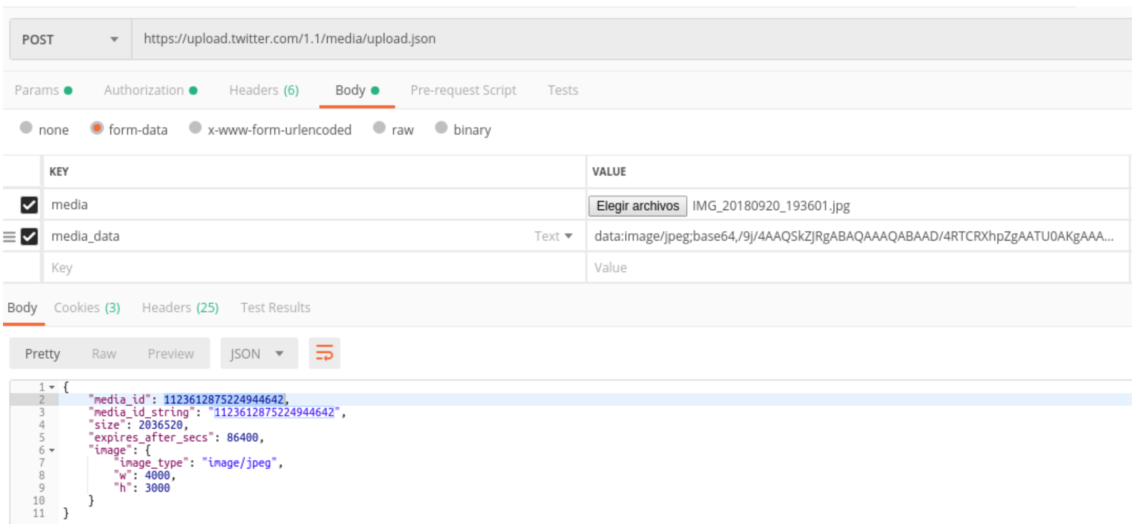


Figura 27. Petición para obtener la ID de la imagen.

Con estos datos, realizamos la misma petición que para publicar texto, pero añadiendo un parámetro más a la URL: **?media="ID"**. En este caso, se nos quedaría la URL de la siguiente manera:

…statuses/update.json?status=M'encanta esta foto&media_ids=1123612875224944642

Y podemos ver el resultado. Se trata de un tweet con el texto "M'encanta esta foto" y una fotografía de mi bicicleta. Además, aunque desde un navegador web Twitter no lo muestra, desde la aplicación móvil para Android podemos ver cómo aparece que el tweet ha sido publicado a través de una aplicación llamada **Capsule_TFG**, que es el nombre que he puesto en la consola de desarrollador de Twitter a mi aplicación. En este caso, se ha querido hacer desde otra cuenta de Twitter, una cuenta privada, para demostrar que cualquier persona que inicie sesión desde mi aplicación con su cuenta de Twitter, va a poder publicar sin problema:

Figura 28. Tweet con la imagen publicado en la cuenta privada @el_xalso donde se puede observar que ha sido publicado a través de una aplicación de terceros.

10.5. Sprint 5: Primera implementación de la interfaz

Se trata de realizar una pequeña implementación básica de la interfaz, puesto que, para pedir acceso a algunas redes sociales, como Facebook, se necesita mostrarles una muestra de la

aplicación, así como un vídeo explicativo como poder acceder a ella, necesitando también disponer de la plataforma web publicada en un servidor y accesible desde internet.

Como ya se ha comentado, se ha realizado con Angular 7, con ayuda para la realización de las interfaces de Angular Material y MDBootstrap.

Las interfaces que se han realizado han sido las que permiten la funcionalidad más básica: "Login", "mis datos", "crear publicación" y "consultar publicaciones". Todo ello con funcionalidad interna, es decir, toda la relacionada con el API Rest propio de Capsule (como por ejemplo guardar los datos al hacer login, guardar las publicaciones, redes sociales que ha sincronizado…); pero de momento no se ha añadido la funcionalidad externa (todo lo relacionado con APIs de terceros), que en el caso de Twitter, sería básicamente implementar las mismas peticiones POST que se han mostrado en el anterior Sprint. Para otras redes como Facebook o Instagram, el funcionamiento sería similar, pero primero sería necesario tener el permiso para publicar, y para ello, estas redes sociales necesitan ver cómo funciona la aplicación (para conceder o no el permiso), y este es el motivo del porqué se ha realizado esta pequeña implementación de parte de la interfaz; que más adelante, se publicará en un servidor web accesible desde internet.

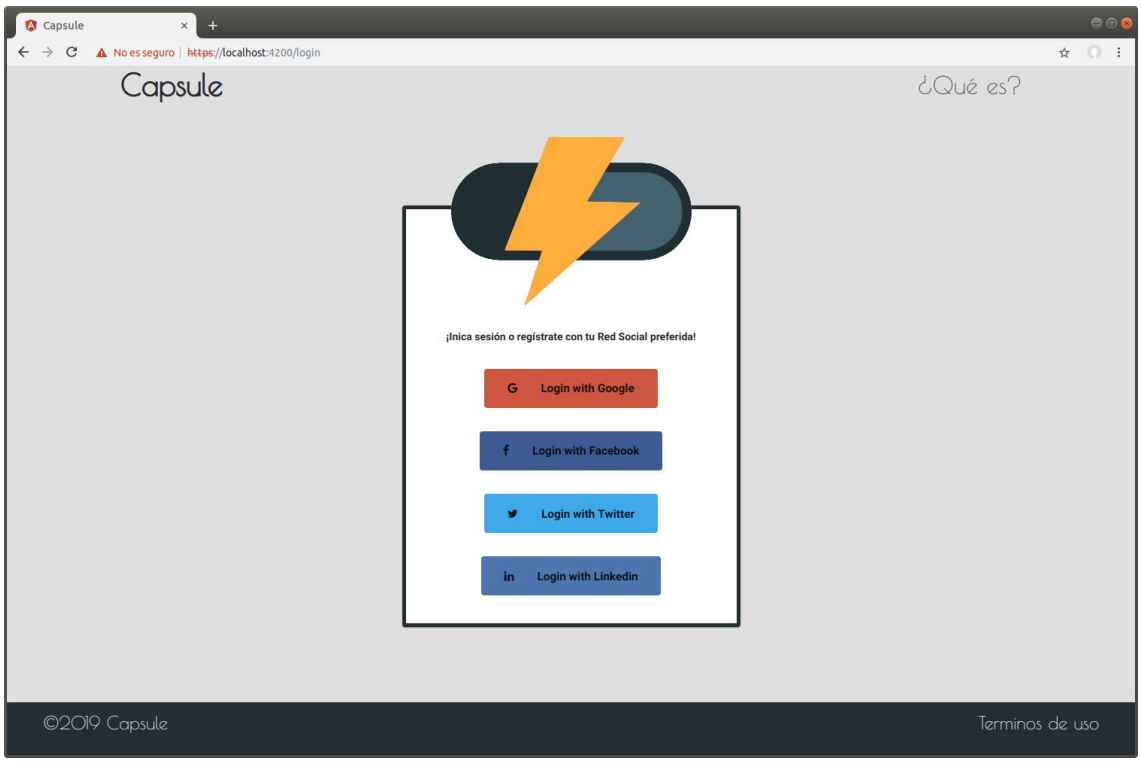


Figura 29. Primera interfaz de la pantalla de Login.

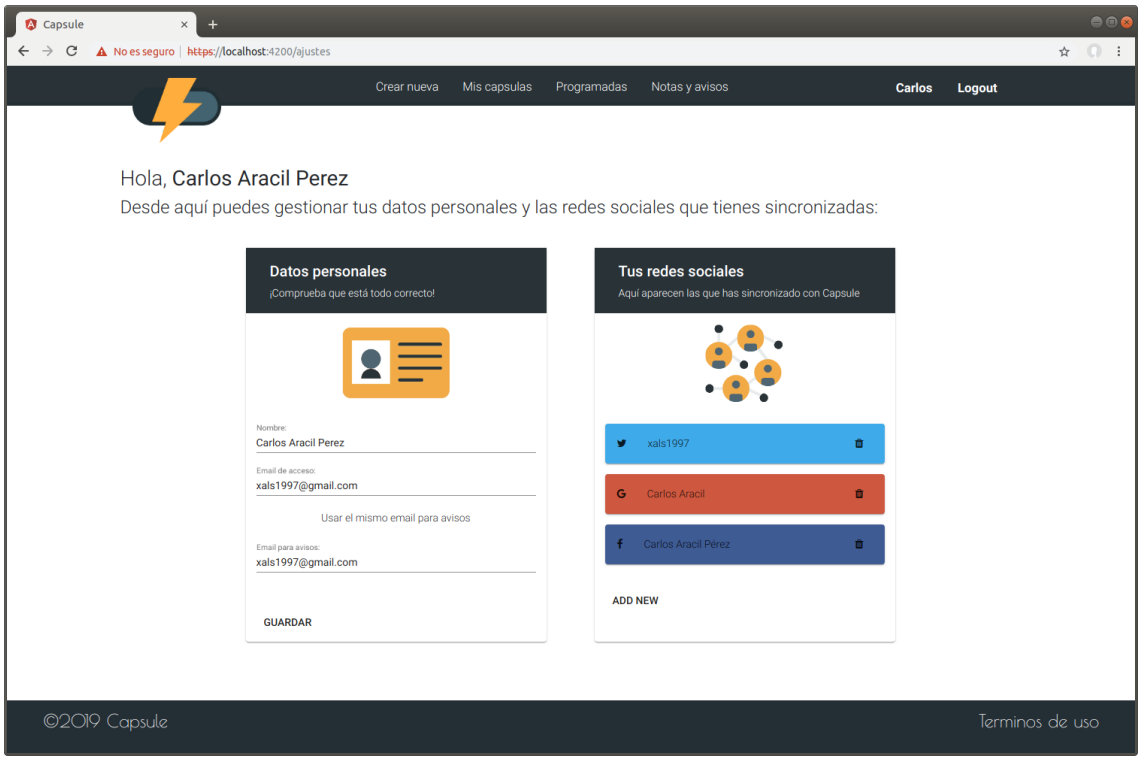


Figura 30. Primera interfaz de los datos de un usuario.

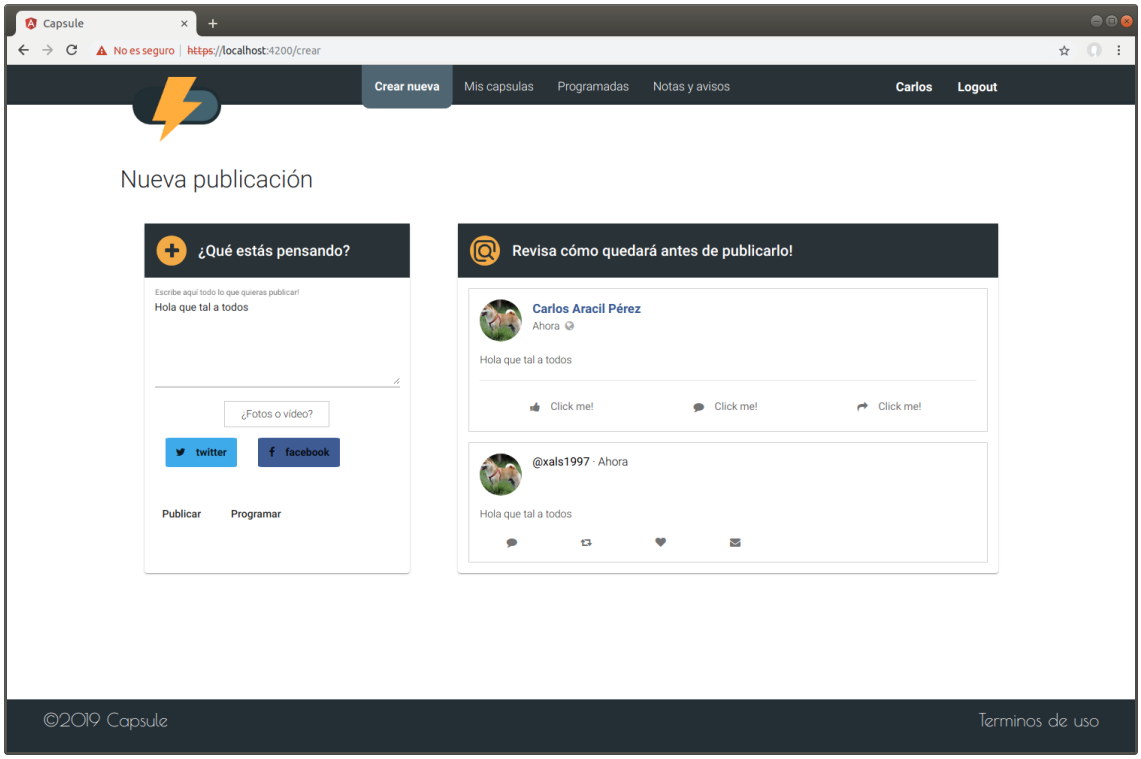


Figura 31. Primera interfaz para crear una publicación nueva. Aparecen únicamente las redes sociales a sincronizadas (en este caso, Twitter y Facebook).

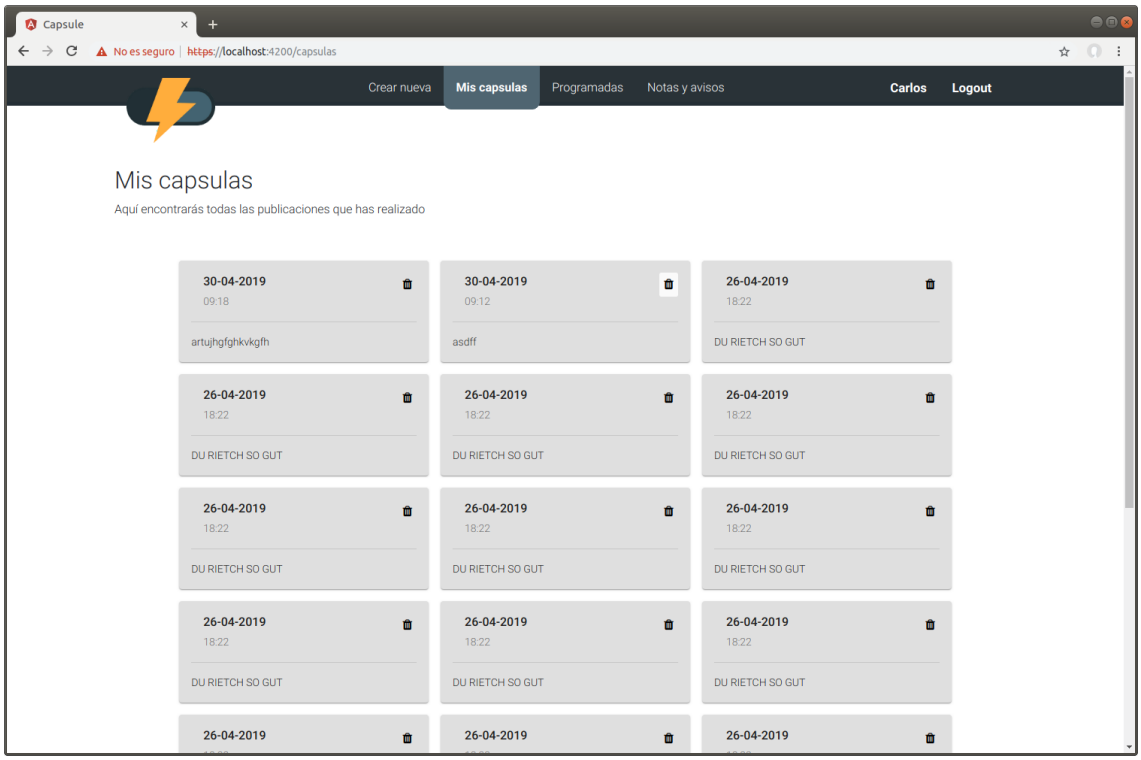


Figura 32. Primera interfaz donde aparecen las publicaciones (capsulas) que se han realizado.

En cuanto a esta última interfaz, se ha implementado lo que se conoce como "Lazy Loading", es decir, conforme el usuario vaya haciendo scroll vertical, irán cargando desde el API Rest, para no devolverlos todos de golpe.

Por defecto, los carga de 10 en 10. Si al cargar 10 todavía no ha habido scroll (en una pantalla grande como en la que se ha probado), entonces volverá a cargar otros 10, hasta que la altura de la página sea superior a la de la ventana (es decir, entonces existirá scroll). Si al obtener los datos del servidor, obtuviera menos de 10, significa que ya no hay más datos que cargar, y por tanto, ya no volvería a llamar a la función que obtiene los datos.

Para llamar a la función, lo hace cuando detecta que la página ha llegado al final, es decir se ha acabado el scroll y el usuario se encuentra en la parte más inferior.

```
if ((window.innerHeight + window.scrollY) >= document.body.scrollHeight)){

}
```

- **window.innerHeight**: altura de la ventana del navegador.
- **window.scrollY:** devuelve el número de píxeles que han sido desplazados en el documento mediante el scroll vertical.

- **element.scrollHeight**: la altura del contenido de un elemento, incluyendo el contenido que no es visible en la pantalla debido al desbordamiento.

Para ello, se suma la altura de la ventana del navegador junto con los pixeles que se han desplazado haciendo scroll, y si eso superara o fuese igual que el tamaño de un elemento concreto (lo podría superar, puesto que scrollHeight no cuenta los márgenes de dicho elemento) entonces habríamos llegado a la parte más inferior del elemento.

10.6. Sprint 6: Configuración del servidor web

Puesto que, para pedir permiso en algunas redes sociales como Facebook o Instagram, uno de los requisitos que solicitan es que la aplicación tiene que estar accesible a internet, es necesario para ello tenerla desplegada en un servidor.

Para ello, se va a utilizar la Raspberry Pi de producción que ya se había mencionado con anterioridad. De momento, la aplicación se encuentra en total desarrollo, y el único motivo por el que se va a proceder a a publicar de forma "temporal" en un servidor accesible a internet es para cumplir con el requisito que pide Facebook y poder conseguir que nos den permiso para poder publicar. Además, no se quería utilizar ningún VPS, ya que, en primer lugar cuestan dinero, y segundo, no se sabe cuánto tiempo tarda Facebook en conceder este permiso. Por este motivo, era preferible buscar alguna alternativa que sirviese igual y no tener que desembolsar dinero. El VPS se podría considerar como una opción cuando la aplicación se encuentre en un estado mucho más maduro y funcional, puesto que entonces sí que tendría sentido, pero de momento, para una simple comprobación por parte de Facebook, no es necesario.

El hacerlo en una Raspberry antes que en cualquier ordenador es por el hecho de que consume muy poco (5V), no hace ruido y es pequeña (se puede esconder detrás de cualquier mueble y ni se ve). La Raspberry podría pasarse meses encendida actuando como servidor y no se notaría en el consumo eléctrico, ni molestaría en cuanto a rudio. El gasto energético es prácticamente similar al de tener un móvil cargando (y de hecho, utilizan el mismo tipo de cable micro USB). Si por el contrario este servidor se configurara en un ordenador personal, "normal", como una torre de sobremesa, entonces estaríamos hablando de tener durante varias semanas o meses 24 horas al día los 7 días de la semana un elemento molesto, ruidoso, grande y que ocupa un espacio considerable, que necesita estar sobre una superficie plana (como una mesa, o en el suelo) y con un consumo eléctrico muchísimo más elevado; cosa que es inviable tener en una casa ya no solo por el gasto, sino por comodidad.

Para configurar el servidor web desde 0, en la Raspberry pi, ha sido necesario llevar a la cabo la realización de las siguientes tareas:

10.6.1 Seguridad en el servidor

Se ha cambiado el puerto por defecto de SSH y se ha introducido autenticación en 2 pasos mediante **Google Authenticator**. (Una vez introducida la contraseña, se pide un código que se obtiene a través de una aplicación en el móvil). Se ha bloqueado el acceso como root por SSH y también se ha implementado **Fail2Ban** para bloquear IPs que intenten realizar más de 6 intentos de inicio de sesión incorrectos.

10.6.2 Apache + MariaDB y MySQL

```
$ sudo apt install apache2
$ sudo systemctl status apache2

$ sudo apt install mariadb-server mariadb-client
$ sudo systemctl status mysql
$ sudo mysql_secure_installation
```

En la configuración de Apache, se ha establecido una redirección permanente del puerto 80 al 443, para acceder siempre a través de **https**.

10.6.3 DNS Dinámica

Para poder acceder desde el exterior, se han abierto los puertos necesarios en el router, en este caso el 80, el 443, y otro para el SSH.

Una vez con los puertos abiertos, es bastante engorroso tener que recordar de la dirección IP asignada por el proveedor de internet, y además, que en caso de reiniciar el router, o si sucediera un corte de luz por algún motivo, esta IP cambiaría, puesto que se trata de una IP dinámica.

Para ello hay dos opciones: o bien contratar una IP fija (cosa que no todos los proveedores de internet permiten, y además supone un precio más elevado), o bien configurar un servicio de DNS dinámicas. En este caso, se va a optar por la segunda opción, y se va a hacer a través de **NO-IP (noip.com).** NO-IP es un servicio de DNS dinámicas, con planes gratuitos y de pago. El plan gratuito ofrece un máximo de 3 dominios, que tienes que renovar cada 30 días y sólo del tipo que ellos ofrecen gratis (como tudominio.onthewifi.com, tudominio.ddns.net, …), es decir, no se va a poder contar con un dominio único (como podría ser capsule.com, simplemente) de forma gratuita; para ello se necesita pagar. Pero para este trabajo, y más para un uso temporal y de pruebas, el plan gratuito es más que suficiente.

Se registrado el siguiente dominio: **capsule.onthewifi.com**, y se ha configurado la Raspberry de manera que, cada 5 minutos, compruebe si la dirección IP ha cambiado. En caso de ser así, un script actualizaría la nueva IP en la base de datos de la plataforma NO-IP.

Así, ya podemos acceder al servidor desde el exterior recordando el dominio anterior.

Prácticamente nos va a servir y se le va a dar el mismo uso que si se hubiera contratado una VPS de las más baratas en alguna plataforma como por ejemplo OVH; pero en este caso, se encuentra en casa y gratis. Para algo temporal como pedir acceso a Facebook, cumple con su función.

10.6.4 Certificado SSL

Para la obtención de un certificado, se ha utilizado **Letsencrypt**, un servicio de certificados SSL gratuitos, que hay que renovar periódicamente.

```
sudo -H ./letsencrypt-auto certonly --standalone -d capsule.onthewifi.com
```

Tras esto, se obtienen una serie de claves que hay que introducir en el archivo de configuración ssl de apache (**/etc/apache2/sites-available/default-ssl.conf**).

10.6.5 Migrar aplicación web

Para ello hay que "compilar" la aplicación Angular en modo producción. Se realiza con el comando **ng build –prod**, que "convierte" y genera toda la aplicación en archivos html, css y javascript; que luego se pueden copiar y pegar en cualquier servidor web.

*Figura 33. Aplicación funcionando ya accesible desde cualquier parte de internet a través del domino **capsule.onthewifi.com**.*

10.7. Sprint 7: Pedir permisos a Facebook

Para poder publicar en nombre de una página en Facebook (recordemos que no permiten publicar en nombre de un usuario, únicamente permiten páginas) necesitamos los permisos **manage_pages** y **publish_pages**. Dentro de la consola de desarrollador de Facebook encontramos muchísimos permisos, que podemos seleccionar para nuestra aplicación. Es este caso, sólo necesito los 2 anteriormente comentados.

Para cada uno de estos permisos, Facebook necesita, además de aceptar su política de privacidad y sus términos y condiciones: tener la web accesible en internet, explicarle los pasos que tiene que realizar para poder realizar la opción para la cual solicito el permiso, explicar por qué se necesita y cómo se va a utilizar el permiso y subir un vídeo mostrando el funcionamiento de este en la web.

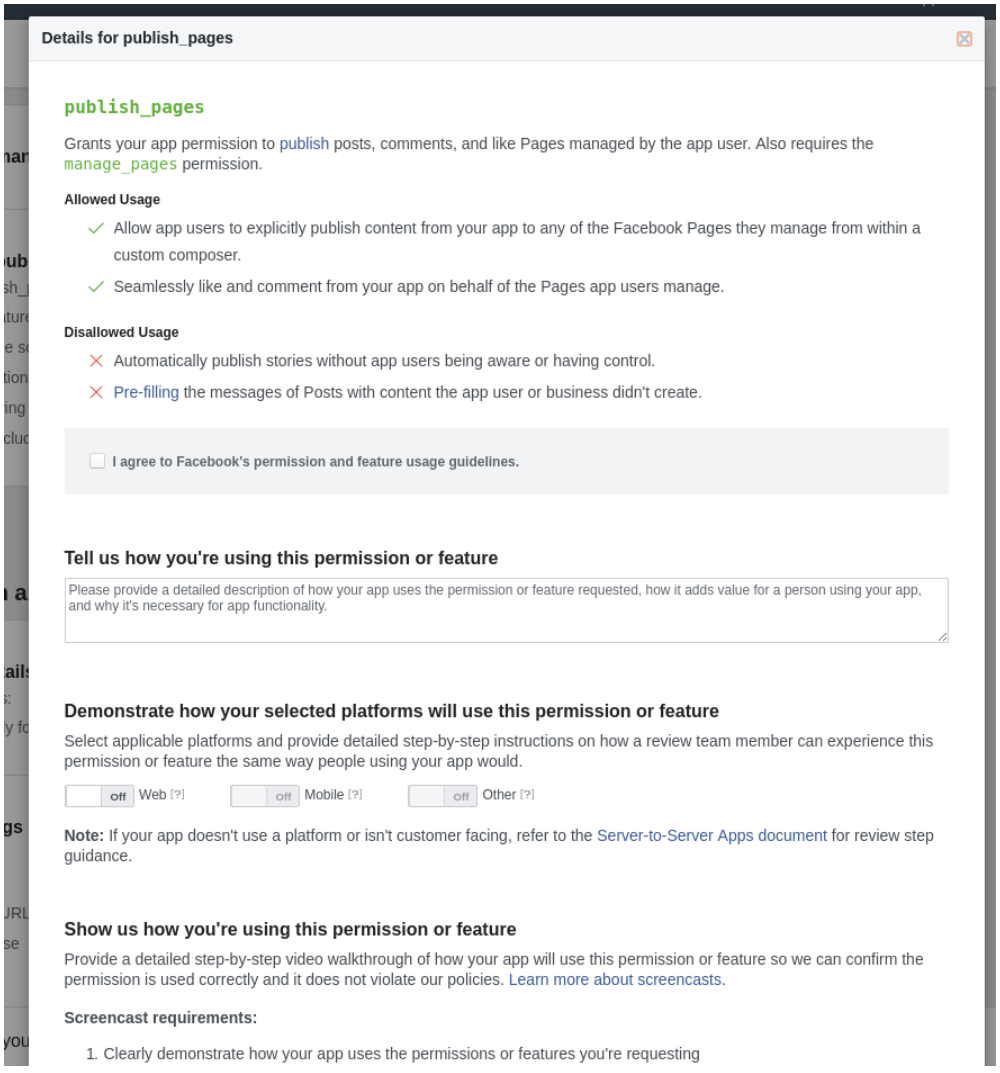


Figura 34. Campos requeridos por Facebook para el permiso publish_pages.

La verdad, es que estos permisos contrastan bastante con los que piden otras redes sociales como Twitter, que en cuyo caso, ha bastado con enviarles un texto explicándoles el motivo y por qué se necesitan los permisos. Y en un par de días estos permisos fueron concedidos.

Una vez rellenados todos los datos que pedían, era necesario completar la información de la aplicación incorporando un icono y un enlace a la política de privacidad.

La URL (**capsule.onthewifi.com**) parece que no acaba de convencer en Facebook, y se marcan como maliciosa, por tanto, no podemos tener ahí la política de privacidad. Esto supone un problema para poder solicitar también los permisos, por lo que se tendría que buscar un dominio más fiable, o contratar un servidor VPS con IP fija y no depender de servicios de DNS dinámico.

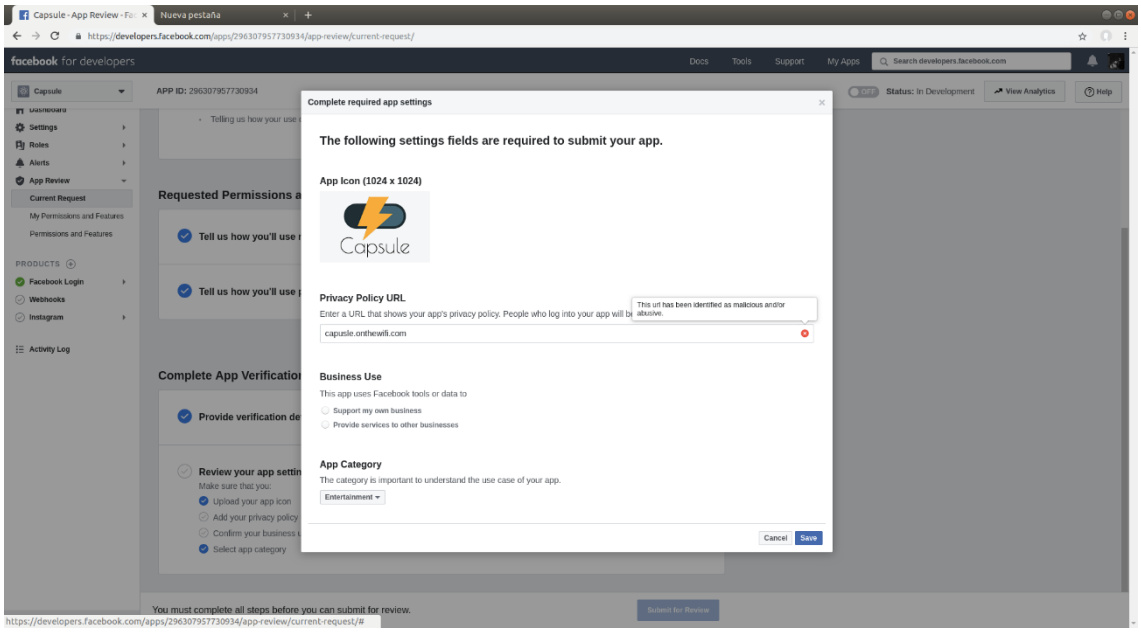


Figura 35. Ejemplo del error con la URL marcada como maliciosa.

Sin embargo, tras esto, y después de configurar todo el servidor, tener ya la web accesible a internet, y disponer de un vídeo grabado mostrando un pequeño funcionamiento de la parte más básica de la aplicación, no era posible pedir permiso a Facebook con nuestro dominio.

Para ello, Facebook dispone de una herramienta de depuración para, analizar entre otras cosas, las urls: https://developers.facebook.com/tools/debug/sharing/

En esta página, se indicaba que el dominio no cumplía con los estándares de la comunidad de Facebook. Sin embargo, existe un botón bien para contactar con ellos en caso de que exista cualquier error o para avisarles de algún problema.

Se ha explicado el motivo y actualmente se está a la espera de disponer del dominio desbloqueado para que no sea considerado como malicioso.

Figura 36. Proceso para solicitar el desbloqueo del dominio

Actualmente nos encontramos a la espera de tener el dominio desbloqueado para poder seguir con el proceso para conseguir los permisos necesarios.

Solicitar este tipo de permisos, es un proceso largo, como se puede ver en la siguiente imagen:

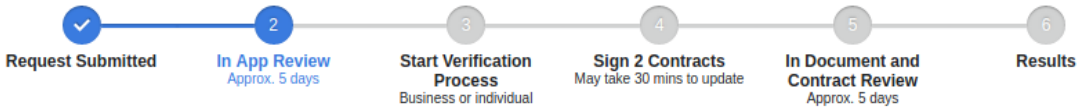


Figura 37. Línea de tiempo para pedir permiso a Facebook.

Son 6 pasos, en los que se incluyen el análisis de la aplicación por parte de Facebook, la verificación, firmar dos contratos y crear una serie de documentos. Y a todo esto, en nuestro caso, tenemos que añadir también el proceso de verificación de la url considerada como maliciosa.

10.8. Sprint 8: Continuar con la integración de Twitter

Ya se vio en el Sprint 4, el cómo crear una publicación desde Postman. Ahora es el momento de introducirla en el código para poder realizarlo desde la aplicación.

Para evitar tener que crear la “firma” de autorización que requiere Twitter para las peticiones de forma manual, existe un paquete instalable para **npm** llamado “twitter”, el cual se encarga de realizar esta operación, y de esta manera, desde nuestro API Rest, podemos realizar la petición que queramos simplemente pasándole los datos del OAuth y la url específica, así nos ahorramos tener que calcular crear a mano las funciones para el hash HMAC-SHA1 y conversiones a base 64.

Una vez integrada la petición POST para crear un tweet, es momento de investigar cómo obtener los tweets enviados.

Al crear un tweet nuevo, en la base de datos se guarda la ID que devuelve Twitter de ese tweet. De esta manera, teniendo ya la ID, simplemente tendríamos que realizar una petición GET a:

https://api.twitter.com/1.1/statuses/show.json

Para esto también es necesario disponer de una “firma”, que obtendremos de la misma manera que para crear un tweet, como “Authorization” en la cabecera de la petición.

Una vez hemos realizado lo anterior, ya obtenemos los datos del tweet: cantidad de retweets, favoritos, la hora a la que se publicó, la id, la id del usuario…

Los datos sobre retweets y favoritos se han incluido en la sección de “Mis Capsulas”. De esta manera, debajo de cada capsula, se puede ver la cantidad de retweets y favoritos que ha tenido:

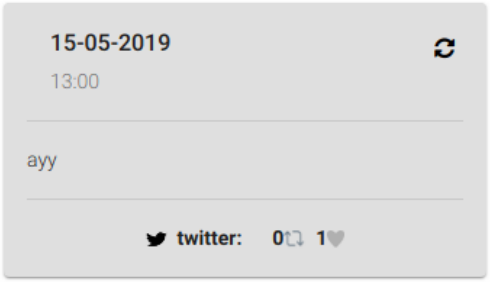


Figura 38. Ejemplo de un tweet que tiene 1 favorito.

Sin embargo, la petición al API de Twitter, como ellos mismos indican en su página para desarrolladores, tiene un límite de 900 peticiones cada 15 minutos. Por tanto, era inviable llamar a este API cada vez que se cargaran las publicaciones, ya que se podría superar el límite en menos

de 15 minutos, o un usuario podría intentar actualizar la página cada poco tiempo para conseguir llegar al límite.

Por tanto, se ha optado por actualizar los datos del tweet cada 15 minutos.

10.8.1. Evitar sobrepasar el límite de la API de Twitter

Para ello, en la base de datos se va a guardar la cantidad de retweets y favoritos, y la hora a la que se han actualizado.

De esta manera, al llamar a la petición, se comprobará la hora que figura en la tabla (la hora de la última actualización) junto con la hora actual. Si no han pasado 15 minutos, no se actualizará.

```
this.currentdate = new Date().toISOString(); //fecha actualizar
if(((Date.parse(this.currentdate) - Date.parse(item.updated)) / 60000)<15){ //comprobar fecha
    console.log("no se puede actualizar aún");
}
else{
```

Figura 39. Comprobar la diferencia en minutos.

currentdate es la fecha y hora actual, y **item.updated** es el nombre de la columna en que se guarda la fecha y hora de la última actualización en la base de datos. Se divide entre 60000 para obtenerlo en minutos. En caso de que el tiempo sea inferior a 15 minutos, no se actualizará nada.

Para comprobar y actualizar el estado del post, se ha incluido un botón, como se puede observar en la figura 38 en el borde superior izquierdo.

Al entrar en la página donde se encuentran todas las publicaciones, para evitar también la sobrecarga, se ha establecido un límite para actualizar automáticamente los posts. Este límite se ha puesto de 10. Es decir, al entrar en la sección, la página actualizará automáticamente las últimas 10 publicaciones, pues se entiende que son las más recientes y en las que el usuario estaría interesado. Para todos los demás, dispone del botón anteriormente mencionado, el cual actualizará el estado actual de esa publicación en concreto.

Tanto al darle al botón de forma manual, como al cargar las últimas 10 publicaciones, se comprobará la diferencia en minutos desde la última actualización. Si han pasado más de 15 minutos, entonces sí se llamará al API de Twitter, se obtendrán los nuevos datos, y se actualizarán en nuestra base de datos.

10.9. Sprint 9: Programar publicaciones

Se va a realizar con un script mediante el uso de **Crontab**. En el sistema operativo Unix, **cron** es un administrador regular de procesos en segundo plano que los ejecuta a intervalos regulares (por ejemplo, cada minuto, día, semana o mes).

Crontab es simplemente un archivo de texto que guarda una lista de comandos a ejecutar en un tiempo especificado por el usuario. Crontab verificará la fecha y hora en que se debe ejecutar el script o el comando, los permisos de ejecución y lo realizará en el background.

Aun así, antes de proceder a realizar el script para publicar las publicaciones programadas, necesitamos saber qué es lo que necesitamos para ello.

En primer lugar, cuando el usuario programa una publicación (una capsula), se guarda dicha capsula con el **texto** y la **fecha** que ha seleccionado. Además, en otra tabla temporal llamada "**pendientes**" se guarda **la ID de la red social** que ha seleccionado, y el token para dicha red social. Es decir, en caso de que el usuario, para una misma publicación (o capsula) seleccione dos redes sociales; dicha capsula estará relacionada con dos elementos de la tabla "pendientes".

El token se guarda puesto que es necesario para poder realizar la publicación. En algunas redes, como Twitter, ese token es permanente siempre que el usuario no desautorice o desconecte la aplicación desde su cuenta de Twitter. En otras redes sociales, como Facebook, los tokens expiran, y es necesario generar uno nuevo.

Para comprobar si un token de Facebook es válido, basta con realizar una petición GET a:

https://graph.facebook.com/me?access_token=x

sustituyendo la x por el token de acceso. En la figura inferior, lo hacemos con un token que no es válido:


```
{
   "error": {
      "message": "Invalid OAuth access token.",
      "type": "OAuthException",
      "code": 190,
      "fbtrace_id": "FhnGeWQ3YYn"
   }
}
```


Figura 40. Ejemplo de respuesta con un token expirado o inválido.

Sin embargo, si realizamos esta misma petición con un token válido, obtenemos nuestro nombre y nuestra ID:

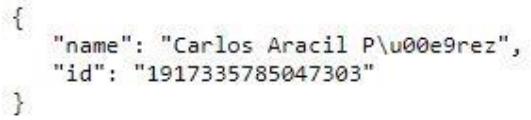


Figura 41. Respuesta con un token válido de Facebook

Este tipo de tokens, los generados al iniciar sesión, son de corta duración o "short-lived tokens", por lo que sólo podríamos programar publicaciones para un tiempo inferior a 2 horas, que es el tiempo que dura un token de corta duración de Facebook.

Para ello, Facebook tiene también los tokens de larga duración o "long-lived", que tienen un tiempo de expiración de 60 días. Para ello, hay que realizar una petición GET y pasar por parámetro **la ID de la aplicación, el secreto de la aplicación y el token del usuario.**

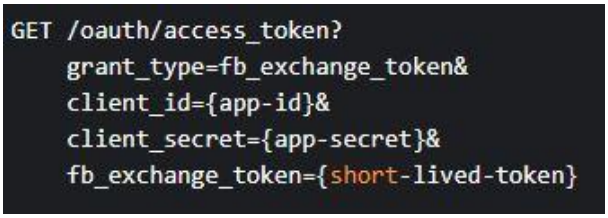


Figura 42. Petición GET para obtener un token de larga duración.

Tras realizar esta petición, Facebook nos devuelve un nuevo token:

Figura 43. Nuevo token generado por Facebook.

Toda la explicación para generar un token de larga duración, se puede encontrar en la página de desarrolladores de Facebook: https://developers.facebook.com/docs/facebook-login/access-tokens/refreshing/

10.9.1 Script y Crontab para publicar las publicaciones programadas

Se ha creado un script que obtiene las capsulas pendientes de programar anteriores a la hora a la que se ejecuta el script.

Con esto, se realiza una consulta SQL para obtener las publicaciones entre las dos fechas.

```
Var date = New Date().toISOString();

SELECT * FROM capsula WHERE f_publicacion < ‘ + date + ’ AND publicada=0;
```

“**publicada**” es simplemente un booleano que se pone como 1 (true) cuando se realiza una publicación, para saber si el contenido se ha publicado ya, y evitar así volverla a publicar. De esta manera, sólo obtiene las capsulas que todavía no se han publicado. Esto sirve también, por si se cayera el servidor, o por algún motivo no se llegara a publicar (por ejemplo, por haber superado el límite del API de la red social), la variable se mantendría a 0 (false), y la siguiente vez que se llamara al script, se volvería a intentar publicar.

Una vez obtengo los resultados, simplemente realizo una petición a la tabla “pendientes” que se ha mencionado ya anteriormente, para obtener a qué red social va dirigida y cuál es el token del usuario.

Tras esto, simplemente falta realizar la misma petición POST que hacemos a la hora de publicar manualmente, pues ya tenemos todos los datos.

Desde Crontab, se ha configurado para que se llame a este script cada minuto:

```
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/1 * * * * /home/ragtime/rest/script.sh >> /home/ragtime/status.log 2>&1
```

Figura 44. Llamada al script desde Crontab. También se guarda el log de la salida del script, para poder observar si ha sucedido cualquier error.

De esta manera, tal y cómo está configurado ahora, el servidor cada minuto comprueba si hay alguna publicación pendiente y entonces, en caso de haber, las publica.

10.10. Sprint 10: Filtros de búsqueda y subida de imágenes

En el apartado de “Mis Capsulas”, donde se encuentran las capsulas ya publicadas, se ha añadido un filtro por fecha, para obtener las anteriores a la fecha indicada y evitar así que el usuario realice un excesivo scroll para llegar a una publicación antigua.

Figura 45. Filtro de búsqueda por fecha y hora.

10.10.1 Publicar imágenes desde la aplicación

Hasta ahora, las únicas pruebas realizadas con subida de fotos habían sido a través de Postman. Para ello, en el backend, se ha utilizado **express-fileupload**, una librería para gestionar la recepción de archivos a través de una petición POST.

A través de esta librería, podemos obtener los archivos separados del body. Es decir, lo enviamos todo junto en el cuerpo de la petición, pero para obtener los archivos realizamos lo siguiente:

```
function(req, res, next){
          req.body --> Variables
          req.files --> Archivos
}
```

Desde el frontend, seleccionamos los archivos (pueden ser varios). Si la red social de destino es twitter, se limita a 4, puesto que Twitter no permite más de 4 en una publicación.

Luego, en el backend, mediante un bucle **for**, obtenemos cada uno de los archivos. Cada uno de estos archivos se envía a Twitter a través de la petición POST ya comentada en el sprint 10.4.2. Para cada uno de ellos, una vez se obtiene la respuesta por parte de Twitter, recibimos una ID. Una vez tengamos todos los archivos con sus respectivas ID, hay que juntar todas estas en un string, que en este caso lo hemos llamado **medias**, donde se irán agrupando, separadas por comas, las ID de cada imagen, ya que así es como lo exige Twitter para enviarlo en la petición de la publicación (para la petición, Twitter necesita un parámetro llamado **media_ids** con el texto “id1,id2,id3”, por ejemplo).

Tras agrupar las ID's que nos ha devuelto twitter, de esta manera, ya tenemos el string con de todas las fotos separadas por comas. Ahora, sólo necesitamos ya el texto de la publicación. Estos dos parámetros son los que se enviarán de nuevo a Twitter (ya como una sola petición) para crear la publicación: el texto + las ID de las fotos que le acompañan.

```
var medias = ""; //CREAMOS EL STRING
for(var i=0; i<req.files.length; i++){ //BUCLE CON TODAS LAS IMÁGENES
        //Petición a Twitter para subir cada imágen:
        client.post('media/upload', {media: req.files[i].data}, function(error, media, response){
                if(media){
                        if(i!=req.files.length-1){ //NO ES LA ÚLTIMA IMÁGEN
                                medias = medias + media.media_id_string+","; //SE AÑADE AL STRING
                        }
                        else{ //LA ÚLTIMA IMÁGEN
                                medias = medias + media.media_id_string; //SIN COMA
                                //Parámetros a enviar a Twitter
                                var status = {
                                        status: body.texto, //EL TEXTO DE LA PUBLICACIÓN
                                        media_ids: medias //EL STRING CON TODAS LAS ID
                                }
                                //Petición POST a Twitter para la subida del post
                                client.post('statuses/update', status, function(err, tweet, resp){
```

Figura 46. Cómo se envía a Twitter una publicación junto con imágenes desde Node

10.10.2 Imágenes en publicaciones programadas

Hasta ahora, cuando se programaba una publicación, únicamente se hacía en modo texto, tal y como se comentó en el sprint anterior.

Sin embargo, a la hora de programar con imágenes, no es tan sencillo. El texto es fácil almacenarlo en la base de datos, y luego, en el momento que se vaya a publicar, se recoge y se envía en la petición. Pero, para las imágenes, necesitamos tenerlas almacenadas en nuestro servidor, para posteriormente recuperarlas y enviarlas a la red social.

Para ello, tenemos en la base de datos la tabla **media**. Esta tabla se almacena la ruta de la imagen y la ID de la capsula a la que pertenece.

Cuando un usuario programa una publicación con imágenes para un futuro, por un lado, se suben las imágenes al servidor, a una carpeta destinada para ello; y por otro, se añaden los datos a la tabla media. El resto de los datos se siguen subiendo a la tabla **pendientes**, como se ha explicado en el sprint anterior. En definitiva, después de que un usuario programa una publicación con una imagen tenemos:

- Datos de la tabla **pendientes:** la ID de la red social, el token (para poder realizar la petición) y la ID de la cápsula a la que pertenecen.
- La imagen subida al servidor en una ruta **/ruta/a/la/imagen.jpg**

- Tabla **capsula** con: su identificador, el texto a publicar, la fecha a la que se publicará y el estado publicada=0.
- Tabla **media** con: la ruta de la imagen y la ID de la capsula a la que pertenece.

Al ejecutar el script para publicar las programadas, que ya se ha explicado en el sprint 9, añadimos una sentencia SQL más, para obtener los datos de la tabla **media** una vez tenemos los datos de la **capsula**. Con la ID de esa capsula, se comprueba si tiene alguna imagen o no en la tabla **media.**

```
sql.query('SELECT * FROM capsula WHERE f_publicacion<="'+date+'" AND publicada=0
        if(results.length!=0){
                sql.query('SELECT * FROM media WHERE ID_c = "'+results.ID+'"', fu
                        if(response.length!=0){} //hay fotos
                        else{} // no hay fotos
```

Figura 47. Comprobar si hay fotos asociadas a una publicación.

En la captura se ha eliminado la parte del código para realizar la publicación (tanto si hay fotos como si no), puesto que se ha mostrado anteriormente: si es sólo texto, se manda como parámetro el texto, y si hay imágenes obtenemos las ID y luego las enviamos junto con el texto.

La única diferencia recae en cómo obtener las imágenes. Cuando se publica al momento, el usuario envía las imágenes al API rest, donde directamente se utilizan en la petición al API de Twitter para obtener las ID's.

En este caso, puesto que no se van a publicar al momento, cuando se envían al API rest, se guardan en una ruta en el servidor (**./Images**), y tras ello, se guardan los datos en la tabla **media**. Tras esto, ya tenemos la foto en la carpeta **Images** de nuestro servidor, y la ruta de la foto y la capsula a la que pertenece guardados en la base de datos:

```
foto : function(req,res){
        let body = req.body;
        if(Object.keys(req.files).length == 0){
                res.status(400).send('No files uploaded.');
        }

        let archivo = req.files.Photo;
        var id = auto_id();

        archivo.mv('./Images/'+req.files.Photo.name, function(err){
                if(err) return res.status(500).send(err);
                sql.query("INSERT INTO media (ID, ID_c, ruta) VALUES ('"+id+"', '"+body.ID_c+"', '"
                        if(error){
                                return next(error);
                        }
                        return res.status(201).send({status: 'ok'});
```

Figura 48. Subida de imágenes al servidor

Luego, a la hora de realizar la publicación, hay que volver a obtener la imagen. De la consulta SQL a la tabla **media** únicamente obtenemos la ruta de la imagen. Para obtener la imagen en sí es necesario hacer uso del módulo **fs** (FileSystem) de node. Este módulo, viene ya instalado por defecto en node:

```
sql.query('SELECT * FROM media WHERE ID_c = "'+results1[0].ID_c+'"', function (error
        //obtenemos datos de la tabla media

        //cargamos la imagen
        var data = require('fs').readFileSync('./Images/'+results[0].ruta);

        //petición a twitter para obtener la id
        client.post('media/upload', {media: data}, function(err,, media, response){
```

Figura 49. Cargar la imagen desde el servidor

Una vez con las imágenes cargadas, obtenemos sus respectivas ID, cargamos el texto a publicar y realizamos la petición a Twitter.

Todo esto se realiza desde **Crontab,** con la llamada al script que realiza cada minuto para cargar el contenido programado.

10.11. Sprint 11: Interfaz para las publicaciones programadas

Se ha realizado una interaz, similar a la de “Mis Capsulas”, pero que en lugar de cargar aquellas que ya se han publicado, en este caso, carga las que están por publicar. Están ordenadas de forma ascendente, es decir: en primer lugar aparecerán las que más cerca están de publicarse. Para cada una, se muestra el día y la hora a la que se publicarán, el texto y a que red social se va a publicar.

Para cargar todas las publicaciones (o futuras capsulas), se utiliza el mismo funcionamiento que en la interfaz de “Mis Capsulas”. Las obtendrá de 10 en 10 conforme se vaya haciendo scroll vertical (“Lazy Loading”), para evitar la sobrecarga en el servidor y dé sensación de fluidez.

También se ha añadido un filtro para poder buscar por fecha futuras publicaciones.

Cada capsula tiene dos botones, uno para editar (cambiar el texto o la fecha) y otro para eliminarla. Al hacer click sobre el botón para editar, se mostrará la pantalla de edición sobre un modal.

Figura 50. Interfaz "Programadas"

Figura 51. Modal para editar una futura publicación.

10.12. Sprint 12: Notas

Se ha realizado la interfaz del último apartado que quedaba de la aplicación: las notas. Al igual que en las interfaces de "Mis Capsulas" y "Programadas", la carga de las notas también se hace con "Lazy Loading" conforme el usuario vaya haciendo scroll.

En la parte superior encontramos la sección para crear una nueva nota y un filtro de búsqueda por fecha.

Al crear una nueva nota, tenemos un campo de texto y un checkbox para activar, si queremos, las notificaciones por email. En caso de marcar este checkbox, tendremos disponible un campo

de fecha y hora para seleccionar a qué hora queremos que nos llegue la notificación al correo. El corro al que se enviarán las notificaciones es el indicado en los ajustes de cada usuario.

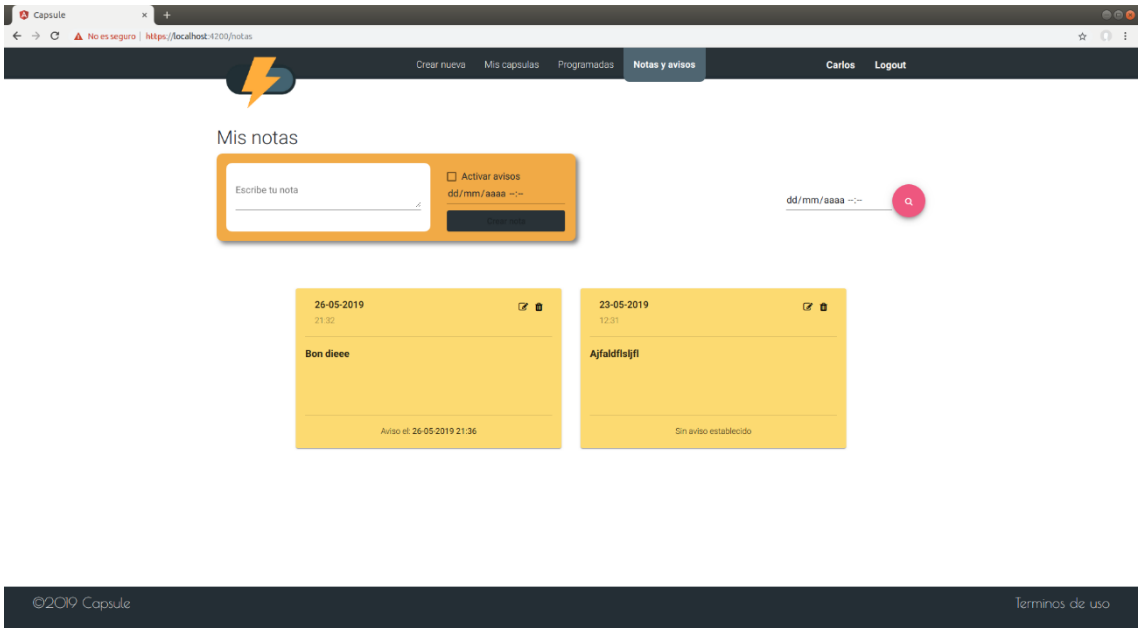


Figura 52. Interfaz de mis Notas.

A las notas se les ha dado un estilo similar al de un post-it. En la parte superior encontramos dos botones: editar y borrar.

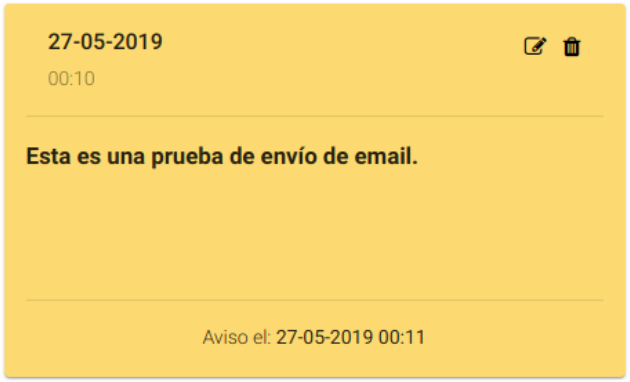


Figura 53. Diseño de una nota, con un estilo similar al de un post-it.

El botón de borrar, con forma de papelera, elimina la nota, mientras que el de editar muestra un modal para modificar la nota: podemos bien cambiar el texto, o activar o desactivar los avisos.

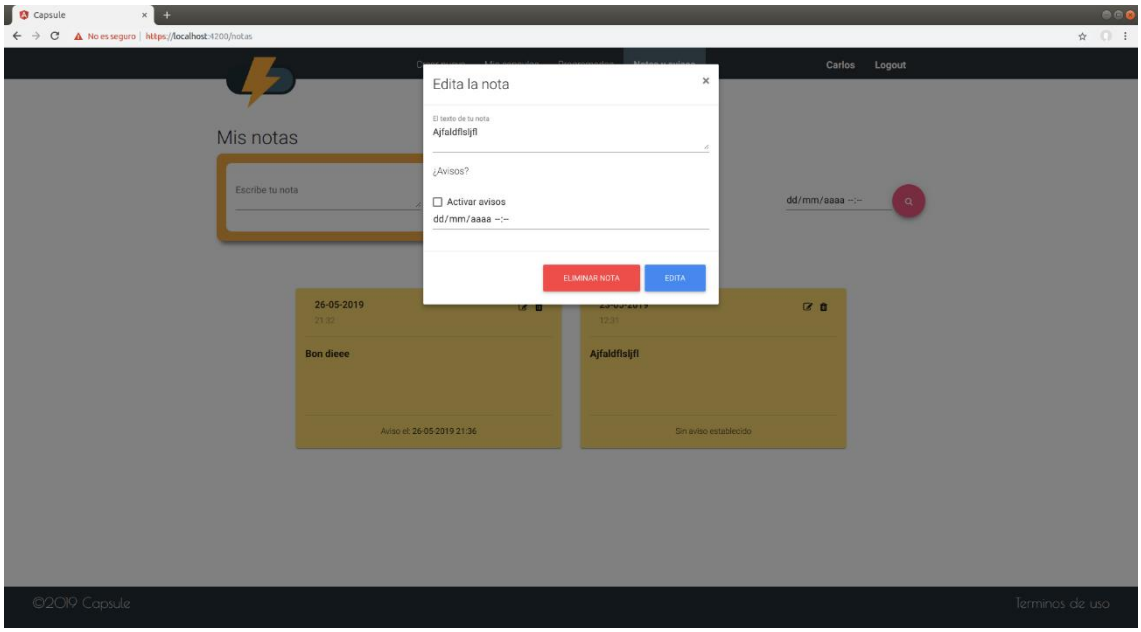


Figura 54. Modal para editar la nota.

10.12.1 Avisos mediante email

Cuando un usuario active el aviso de una nota, se le notificará a través de un email en la fecha y hora seleccionada.

Para ello, se ha creado un script, el cual se ejecutará de la misma manera que el comentado en el sprint 10.9 para publicar publicaciones programadas, es decir: desde Crontab.

Crontab ejecutará el script cada minuto. El funcionamiento del script es el siguiente:

Se selecciona de la tabla **notas** las filas con el campo **aviso** establecido a la hora igual o anterior a la de la ejecución del script, junto con el email establecido para los avisos en la tabla usuario:

```
SELECT notas.texto usuario.aviso FROM notas, usuario WHERE notas.ID_u =
usuario.ID AND notas.aviso<=”’+date+’” ORDER BY notas.aviso DESC;
```

Para cada una de las notas que obtenemos con la anterior consulta, se envía un email con el texto **notas.texto** al correo **usuario.aviso**.

Para enviar el correo desde una cuenta de Gmail creada para a aplicación, hacemos uso del módulo para node **gmail-send**, que se encarga del envío de correos.

Para que el envío funcione, es necesario activar el acceso a aplicaciones menos seguras en la cuenta de Gmail que vamos a utilizar. Esto es necesario activarlo, puesto que, sin ello, Google no permite el envío de emails de forma automatizada.

Figura 55. Permitiendo el acceso a aplicaciones menos seguras en Gmail.

Una vez hemos activado esta opción, el script que hemos creado ya puede empezar a enviar correos desde la aplicación.

En la siguiente figura, se muestra el aviso que se ha recibido en el correo de un usuario, para la nota creada por el mismo, que se ha mostrado de ejemplo en la Figura 54. Aquí se ve el correo recibido:

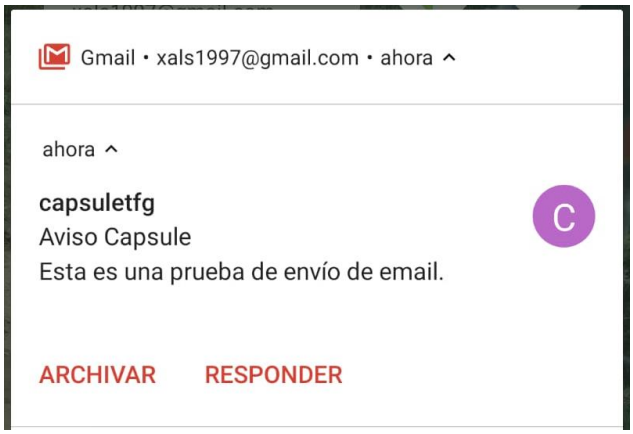


Figura 56. Recepción del email.

10.13. Sprint 13: Consejos al usuario

Como ya se ha explicado, Capsule no sólo gestiona las redes sociales de los usuarios; también ayuda a que las publicaciones lleguen a un mayor número de gente.

Por este motivo, se ha incluido un "spinner" que informará al usuario de la calidad de sus publicaciones.

Dependiendo del contenido de la publicación del usuario, el spinner irá poniéndose rojo, en caso de no hacer caso de las sugerencias; y verde cuando las va cumpliendo.

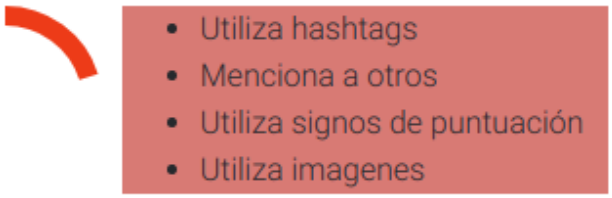


Figura 57. Ejemplos del spinner de "calidad" de la publicación.

Este elemento ha sido colocado en la esquina superior derecha, en la pantalla de crear una nueva publicación:

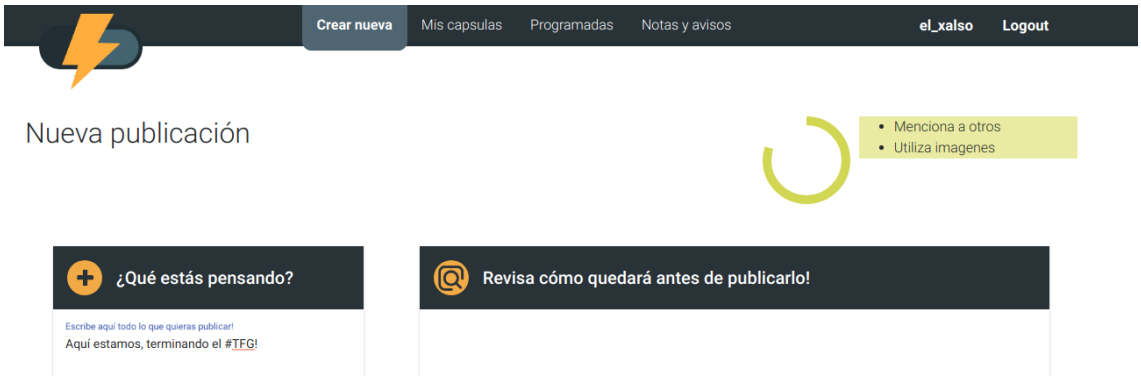


Figura 58. Recorte de la interfaz con el "spinner".

10.14. Sprint 14: Finalizar configuración Back-end

En este último sprint, se ha puesto en producción la versión actual de la aplicación. Para migrar la aplicación de Angular al servidor Apache en la Raspberry pi de desarrollo, se ha realizado tal y como se ha explicado en es Sprint 10.6.5.

Para evitar ataques y accesos no autorizados al servidor se han llevado a cabo una serie de medidas:

10.14.1 Copias de seguridad de la base de datos automáticas.

Se ha utilizado la herramienta **Percona XtraBackup** para realizar copias de seguridad incrementales de la base de datos.

Percona XtraBackup es una herramienta de código abierto para realizar copias de seguridad en servidores que utilizan MySQL; y, además, no bloquea la base de datos mientras se realiza la copia de seguridad. Percona XtraBackup se encuentra integrado en MariaDB, haciendo uso del comando **mariabackup**.

Para realizar las copias, se ha utilizado la herramienta Crontab. Una vez a la semana (el domingo a las 3 de la mañana) se realiza una copia total de la base de datos:

```
0 3 * * 0 mariabackup --backup --target-dir /home/pi/db_backup --user "" --password ""
```

El resto de días de la semana a las 4 de la mañana, se realiza una copia incremental, comparando los datos con la anterior incremental y la última completa:

```
0 4 * * * mariabackup --backup --target-dir /home/pi/incremental/ --incremental-basedir /home/pi/db_backup/ --user "" --password ""
```

Para guardar estas copias de seguridad, en Crontab, después de realizarlas, genero un zip y las envío a través de **scp** a la Raspberry de pruebas, ya que esta no se encuentra abierta a internet.

El zip lo genero a las 4:30 de la mañana, y lo envío a la otra Raspberry a las 5 de la mañana, dando este margen de media hora por si hubiera muchos datos y tardara en generarse el zip. Los domingos, a las 4 de la mañana realizo lo mismo, pero con la copia completa.

```
0 4 * * 0 zip -r full.zip /home/pi/db_backup
30 4 * * * zip -r incremental.zip /home/pi/incremental/
30 4 * * 0 /usr/bin/scp -i /home/pi/pruebas_key -r /home/pi/full.zip
pi@10.0.0.3:/home/pi
0 5 * * * /usr/bin/scp -i /home/pi/ pruebas_key -r /home/pi/incremental.zip
pi@10.0.0.3:/home/pi
```

La dirección 10.0.0.3 es la dirección de la Raspberry pi de pruebas en mi red interna.

10.14.1 Headers de seguridad.

Las cabeceras de seguridad son una parte fundamental para la seguridad de un sitio web. Cuando un usuario visita un sitio a través de su navegador, el servidor responde con una cabecera HTTP. Estas cabeceras le dicen al navegador cómo comportarse durante la comunicación con el sitio.

Gracias a las cabeceras, podemos evitar numerosos tipos de ataques:

- **Strict-Transport-Security**: forzar el uso de SSL.
- **X-Frame-Options:** dice al navegador si el sitio web puede cargarse dentro de un **frame** o no. Evitando que pueda realizarse esto, podemos prevenir ataques como el clickjacking.
- **X-Xss-Protection:** este filtro no permite que la página se cargue cuando detecta un ataque de scripts de otros sitios.
- **X-Content-Type-Options:** protección frente al "sniffing" MIME y obliga al navegador a manter el "Content-type" declarado.
- **Feature-Policy:** establece las API a las que el sitio puede acceder o modificar el comportamiento predeterminado del navegador para ciertas características.
- **Referrer-Policy: same-origin:** solo enviará el origen cuando el destino y el origen sean el mismo.
- **Content-Security-Policy:** es una medida efectiva para proteger el sitio de ataques XSS. Al incluir en la lista blanca las fuentes de contenido aprobado, puede evitar que el navegador cargue activos maliciosos. Establecido **script-src, img-src, object-src…** como **'self'** sólo cargará el contenido del mismo dominio.

Estas cabeceras se incluyen en el documento de configuración de apache, que en este caso se encuentra en **/etc/apache2/sites-avaliable/default-ssl.conf**

```
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-Xss-Protection "1; mode=block"
Header always set X-Content-Type-Options "nosniff"
Header always set Feature-Policy "microphone 'none'; payment 'none'; sync-xhr 'self' https://capsule.onthewifi.com"
Header always set Referrer-Policy "same-origin"
Header set Access-Control-Allow-Origin: *
Header always set Content-Security-Policy: "default-src https: 'unsafe-inline'; img-src *; object-src 'none'"


        </VirtualHost>
</IfModule>
```

Figura 59. Headers en la configuración de Apache.

10.14.1 Creación de una VPN.

Aunque ya se tenía implementada la seguridad para el acceso SSH, se ha cerrado el puerto de acceso desde el exterior, para sólo poder acceder por SSH desde la red interna.

Para ello, se ha creado una VPN, desde la cual, desde cualquier lugar, podría acceder a esta red interna. El porqué de esto es simplemente por aprender y saber cómo configurar una VPN en la Raspberry pi.

Una vez nos encontramos en la red interna, entonces podemos acceder a la Raspberry a través de SSH, y de esta manera, evitamos cualquier tipo de ataque a través de este protocolo.

Gracias a la VPN también puedo acceder en cualquier momento y desde cualquier lugar a la Raspberry de pruebas, así como al router de mi casa, pudiendo abrir o cerrar los puertos si lo necesito (siempre que no cierre el puerto del VPN).

La información sobre la creación de esta VPN se encuentra en el Anexo I, ya que era demasiado extensa como para incluirla en este apartado.

11. Pruebas y validación

Se ha comprobado el correcto funcionamiento de la seguridad de las cabeceras HTTP mediante la página https://securityheaders.com. Esta página, comprueba las cabeceras y nos puntúa:

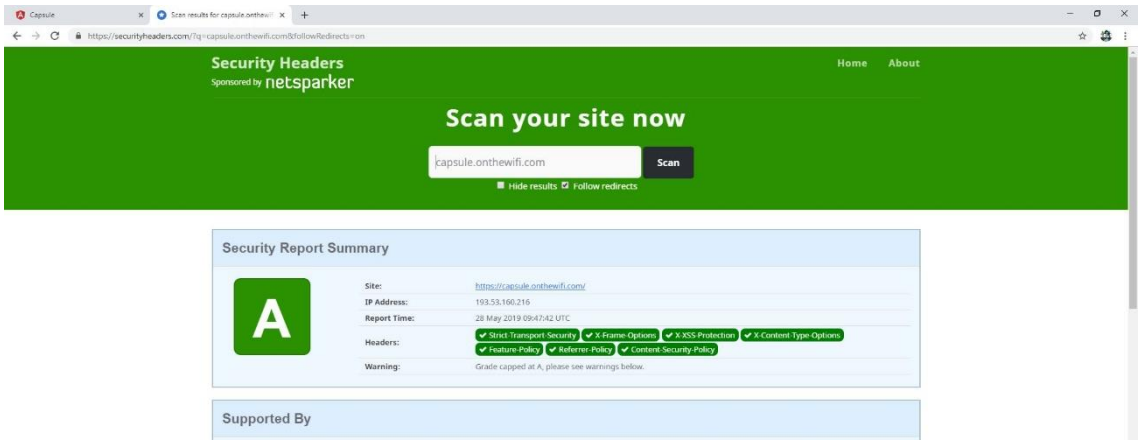


Figura 60. Puntuación "A" en cuanto a seguridad en las cabeceras HTTP.

También se ha analizado la velocidad de carga de la página a través de la aplicación web de Google **PageSpeed Insights**.

Para un ordenador de escritorio hemos obtenido una puntuación de 89%.

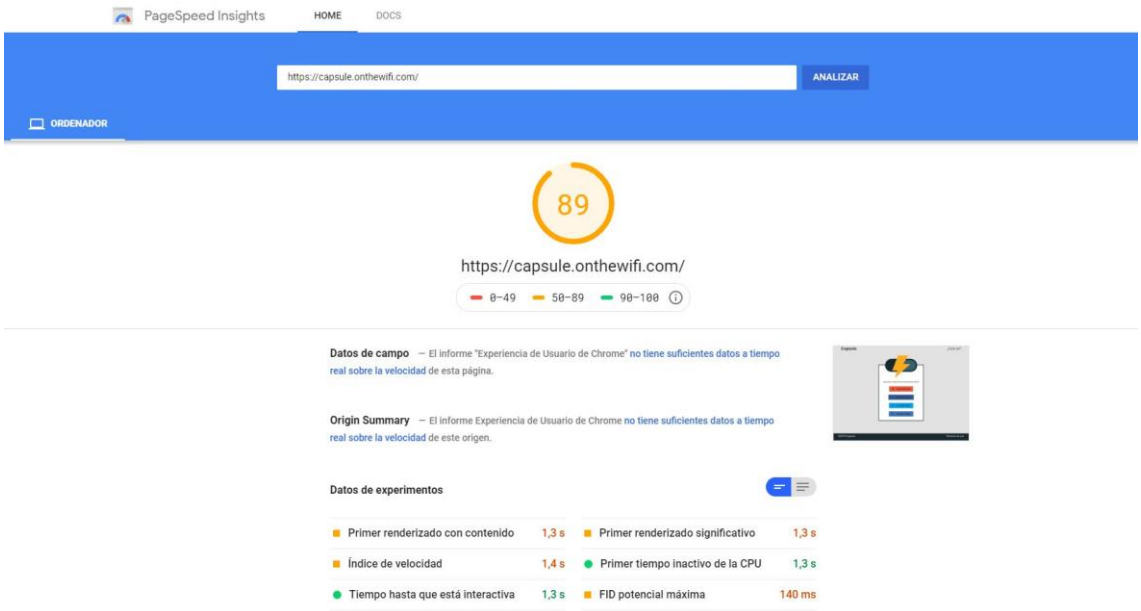


Figura 61. Puntuación PageSpeed Insights

En cuanto a las soluciones planteadas en el Estudio de viabilidad, se han cumplido todas, excepto el apartado de “publicar en distintas redes social”, pues a día de hoy, 28 de Mayo de 2019, ni Facebook ni Instagram han dado permiso para poder hacer uso de su API.

Aun así, teniendo en cuenta que el funcionamiento de la aplicación sería el mismo, simplemente sería añadir nuevas peticiones POST en el backend, se ha procedido a comparar el tiempo que se emplearía en publicar una misma publicación en dos redes sociales (Facebook y Twitter) con el que se tardaría en hacerlo de forma manual:

Publicación en Facebook y Twitter con el texto “Hola que tal #TFG” desde **Capsule**	Tiempo: 9 segundos
“Hola que tal #TFG” de forma individual en cada red social.	Tiempo: 15 segundos.
Publicación en Facebook y Twitter con el texto “una foto” y una imagen adjuntada desde **Capsule**	Tiempo: 12 segundos.
Texto “una foto” y una imagen adjuntada de forma individual en cada red social.	Tiempo: 20 segundos.

Tabla 4. Comparaciones de tiempo entre publicar en Capsule o en cada red de forma separada.

Se puede observar, que en Capsule el tiempo es siempre inferior. Cuantas más redes sociales, o más contenido tenga la publicación (imágenes, hashtags, etc) mayor será la diferencia de tiempo entre publicar desde Capsule y hacerlo en cada red social por separado.

12. Resultados

En esta sección se va a analizar el producto final y mostrar los resultados de cada interfaz:

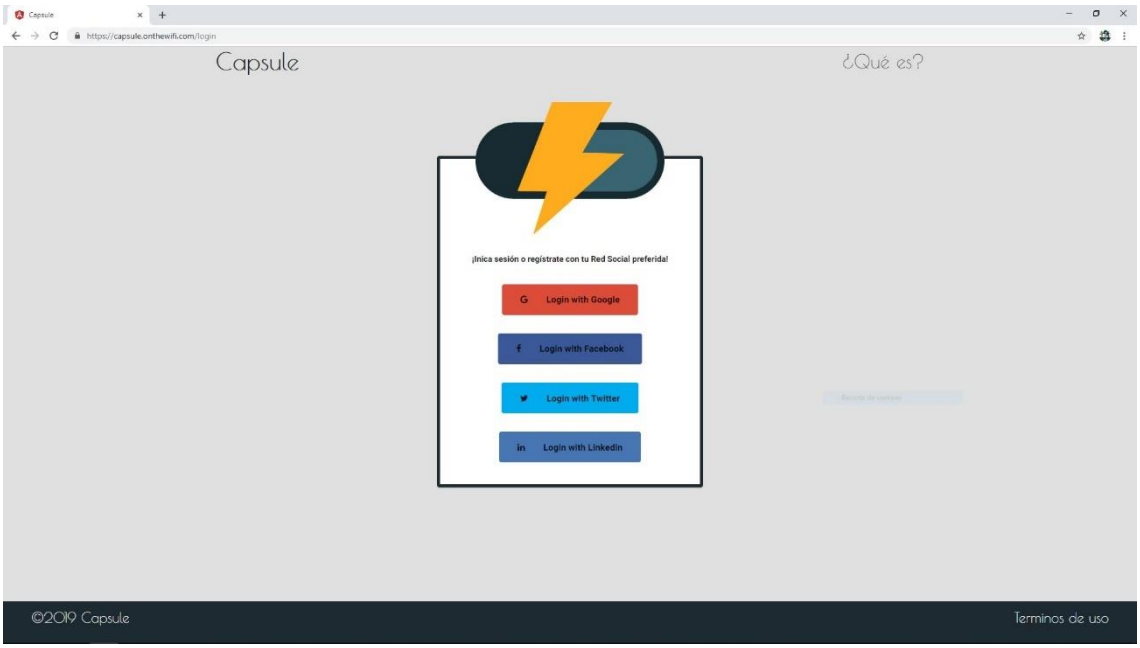


Figura 62. Login

Seleccionamos la red social preferida y se abrirá la ventana para introducir nuestros datos:

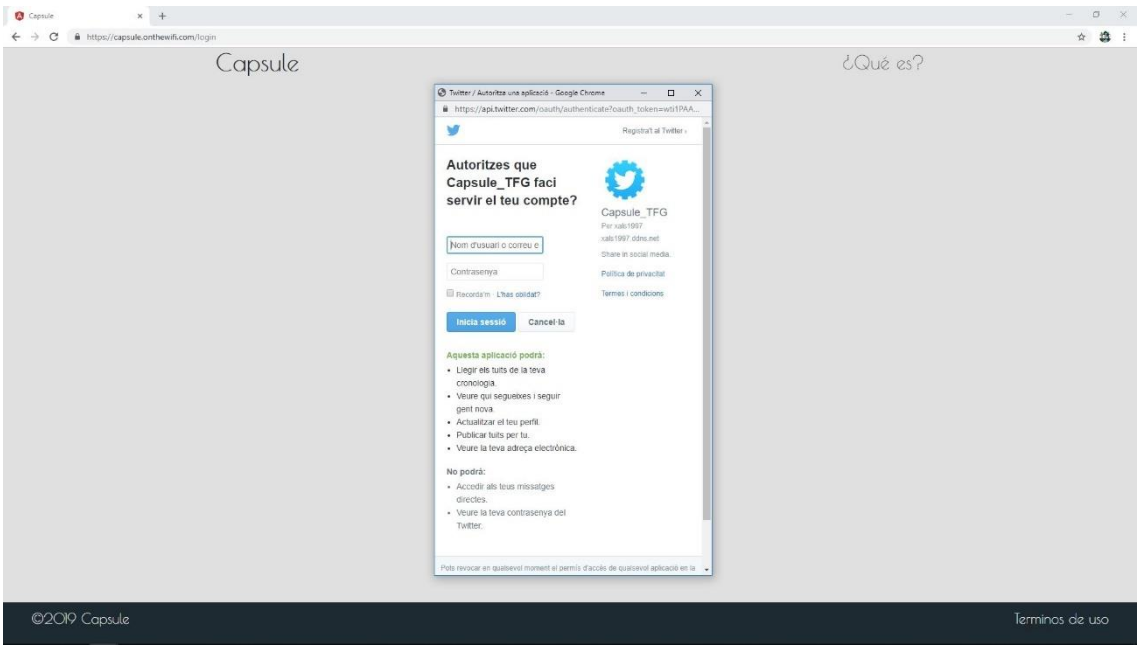


Figura 63. Iniciar sesión a través de OAuth.

Si es la primera vez que accedemos, deberemos aceptar las condiciones:

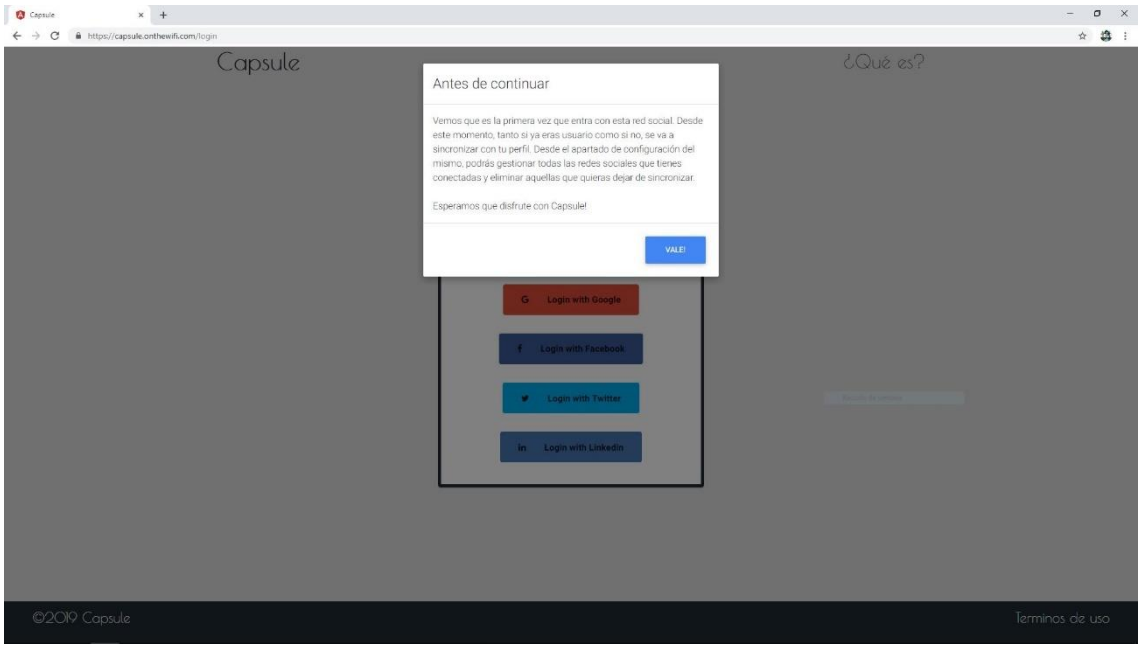


Figura 64. Aceptar condiciones.

Seremos redirigidos a la página para crear nuevas publicaciones, donde, además, según el contenido de la publicación, la aplicación nos valorará la calidad de la misma (esquina superior derecha). También disponemos de una previsualización de cómo quedará:

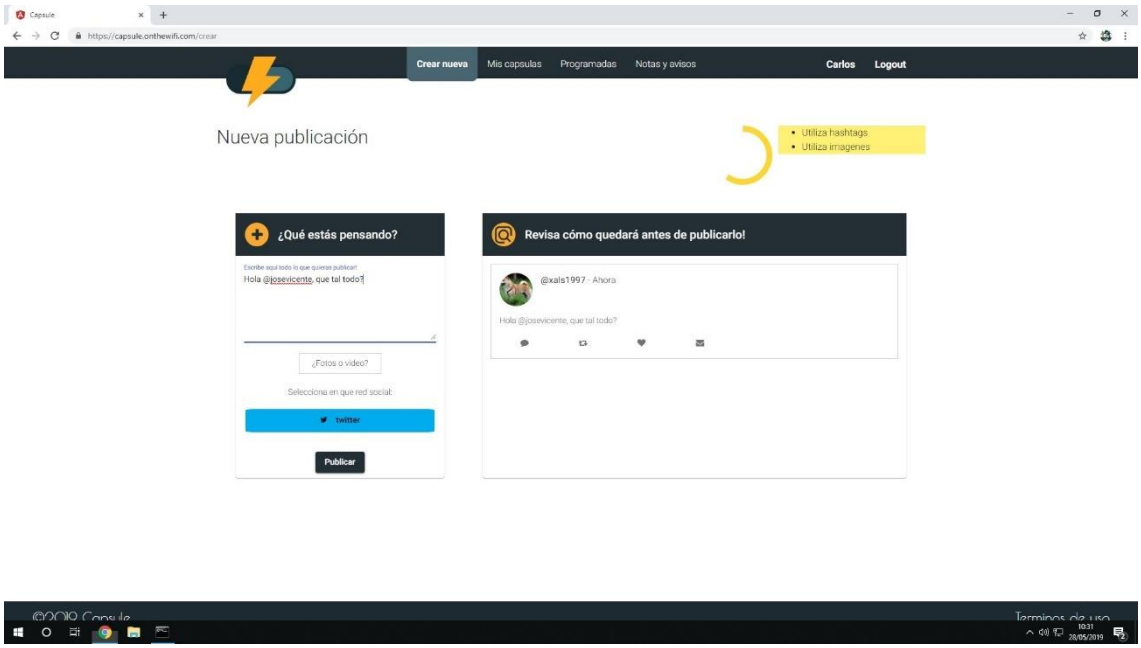


Figura 65. Crear un nuevo post.

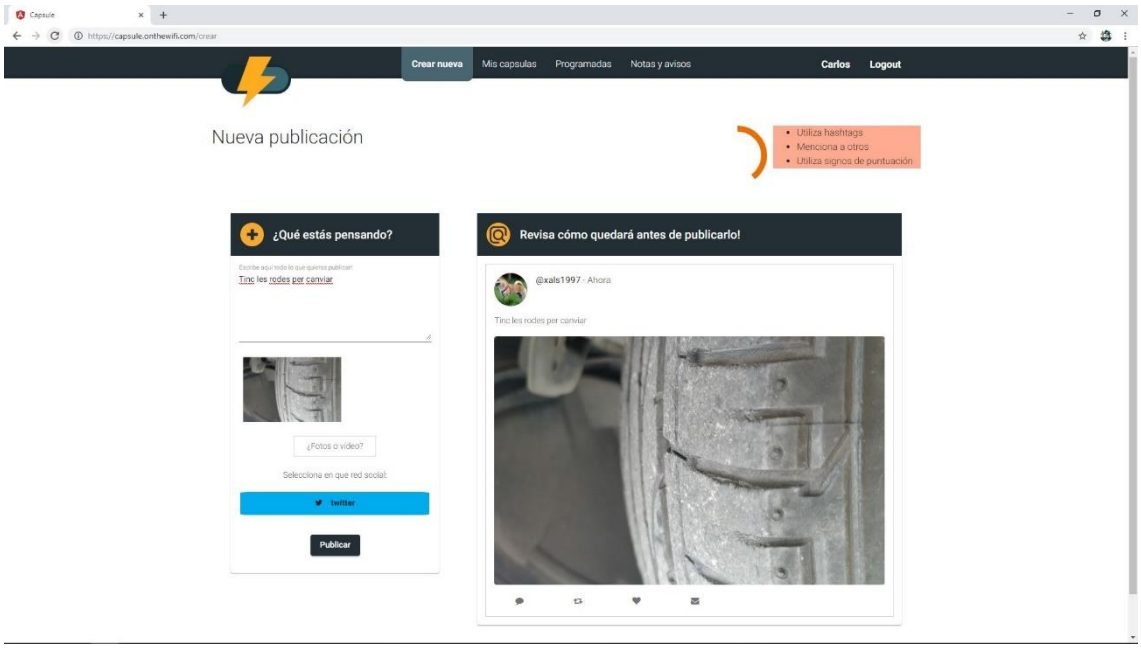


Figura 66. Crear un nuevo post con imágenes.

Aparecerá un modal para decidir si queremos publicarlo ya, o seleccionar una fecha para que se publique en el futuro:

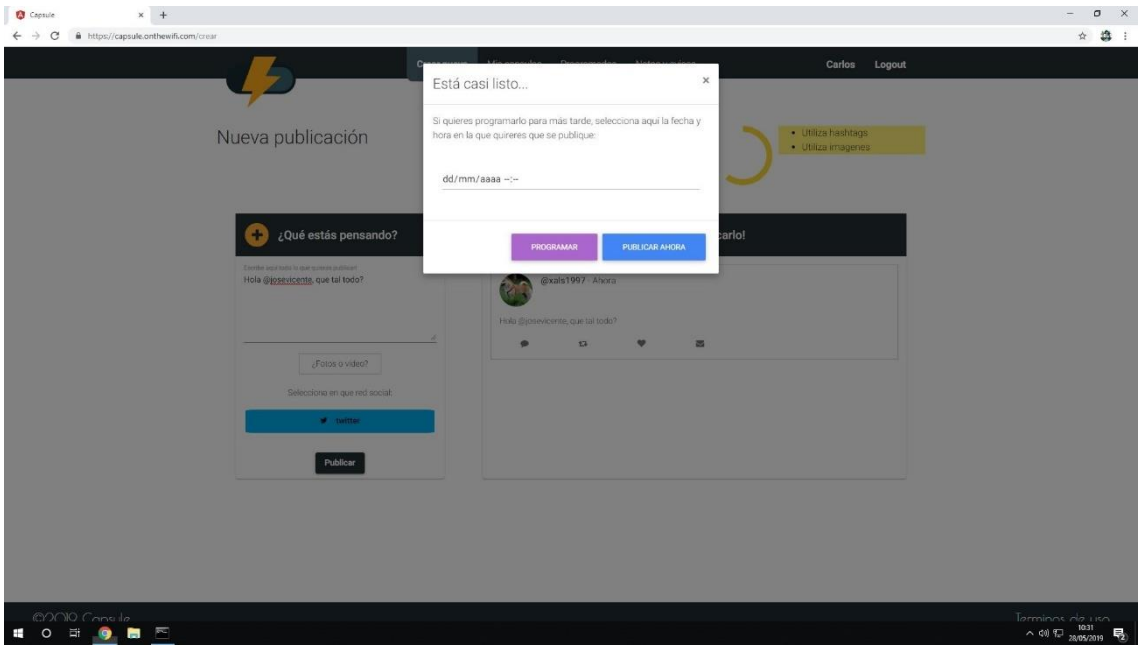


Figura 67. Modal para seleccionar si publicar ahora o programarlo para después.

Una vez publicado, en la sección de “Mis Capsulas” encontramos las publicaciones que ya se han realizado con los datos de interacción de otros usuarios (likes, retweets…):

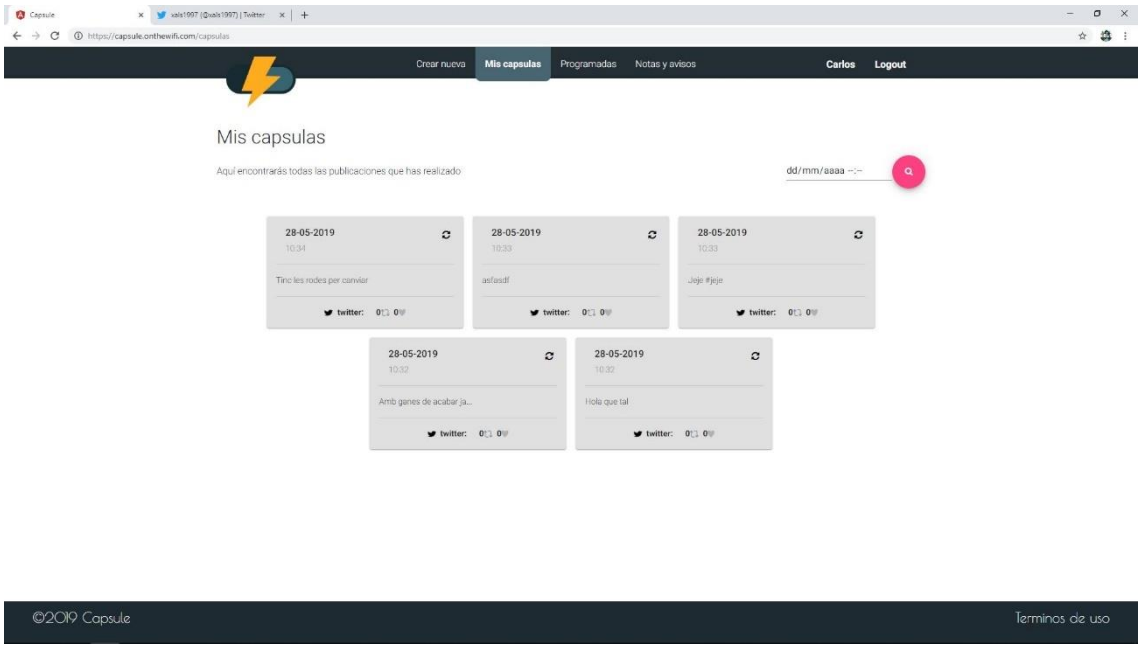


Figura 68. Interfaz “Mis Capsulas” para ver todas las publicaciones realizadas desde la aplicación.

En el apartado “Programadas”, similar en cuanto a diseño al anterior, encontramos también publicaciones, pero en este caso, las que todavía no se han publicado. Son las programadas para un futuro, es decir, las que lo harán en la fecha y hora indicada:

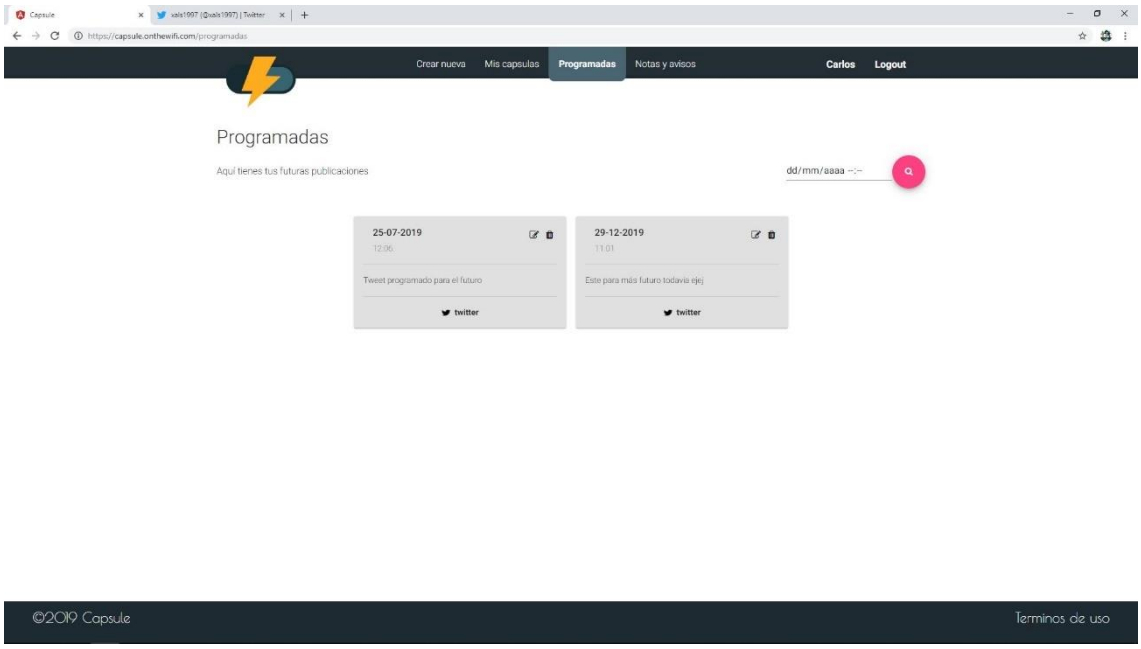


Figura 69. Interfaz de las programadas para un futuro.

Podemos editarlas, cambiando el contenido o la fecha, o eliminarlas:

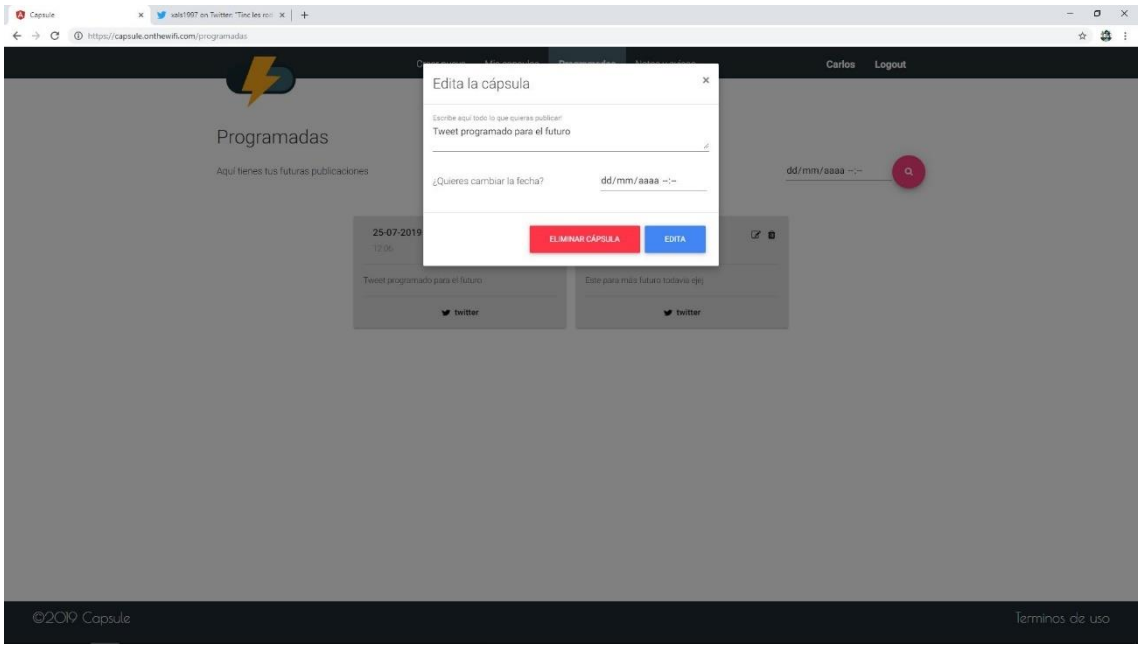


Figura 70. Editar publicación futura.

En la sección de “Notas y avisos”, podemos guardar recordatorios y seleccionar, si queremos, ser avisados a través de un correo de dicho recordatorio (en una fecha y hora indicada):

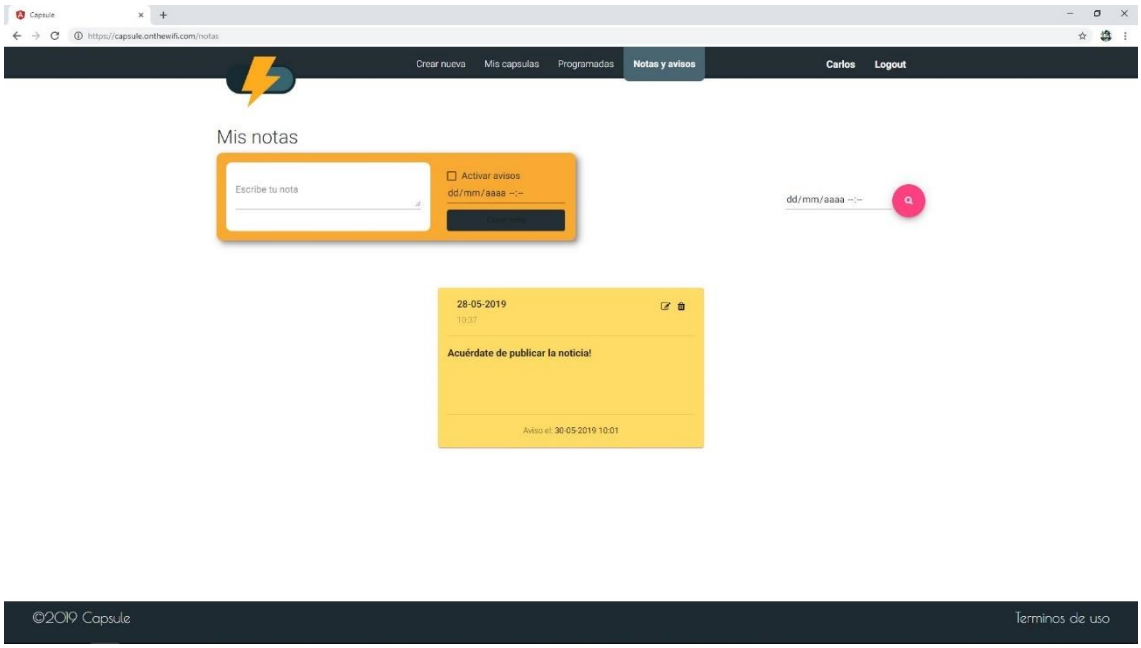
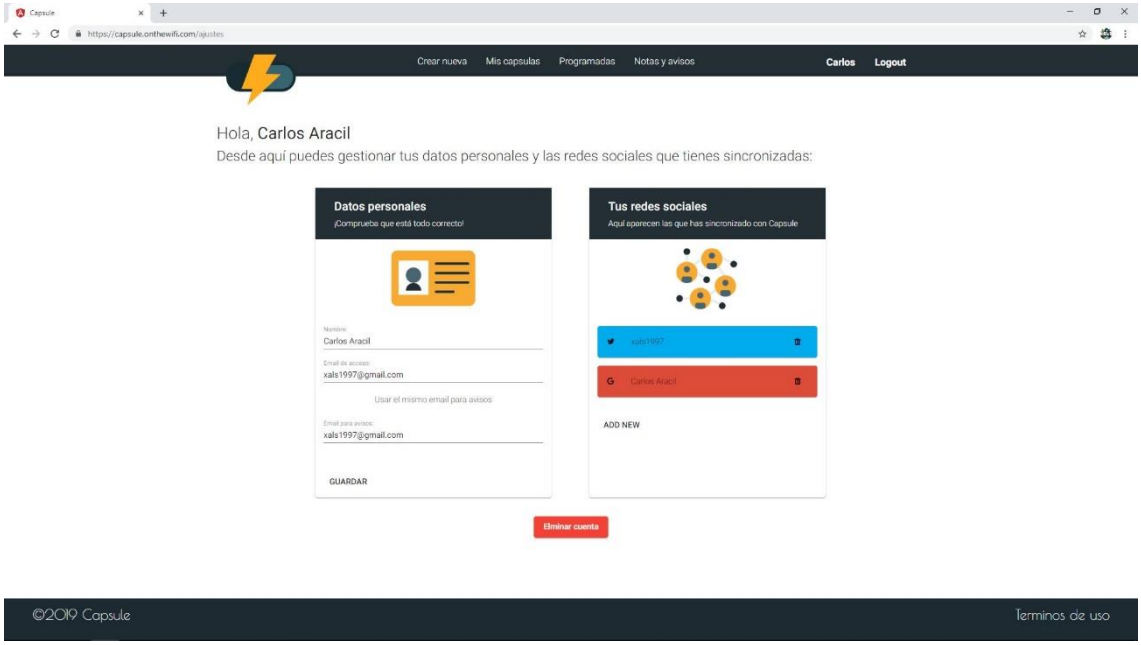


Figura 71. Interfaza “Notas y avisos”.

Por último, si hacemos click sobre nuestro nombre de usuario en el header, accedemos a la sección de ajustes, desde donde podemos editar nuestros datos o eliminar nuestra cuenta:



13. Conclusiones y trabajo futuro

En este apartado se recogen las conclusiones extraídas a lo largo de todo el desarrollo del trabajo.

13.1. Estado actual

Podríamos considerar la plataforma web como un prototipo acabado, puesto que se han cumplido con todos los requisitos funcionales establecidos como “Obligatorios”. Aunque no de un prototipo al 100%, puesto que de los “opcionales”, han faltado los relacionados con el administrador.

En cuanto a las redes sociales principales definidas en un principio, Facebook, Twitter e Instagram, actualmente la publicación sólo funciona con Twitter, ya que tanto para Facebook como Instagram (ambas propiedad de Facebook), se requiere una comprobación y verificación de la aplicación por parte de cada una de ellas para conceder los permisos necesarios para poder realizar la petición POST para publicar.

Sin embargo, el diseño de la aplicación se ha realizado teniendo en cuenta que, en un futuro, podrían conceder dicho acceso. Por tanto, la parte a añadir se realizaría en el API rest, dejando la aplicación prácticamente intacta (únicamente habría que añadir la llamada al API rest con los datos de la publicación).

13.2. Mejoras y ampliaciones

En primer lugar, las ya comentadas: un panel para el administrador, que no se ha podido implementar por falta de tiempo; y establecer la funcionalidad para más redes sociales.

- Login con Linkedin: Funcionando hasta finales de abril. Hasta esa fecha, para el login con Linkedin se utilizaba la librería **angularx-social-login**, que permitía hacer login con Google, Facebook y Linkedin. Sin embargo, esta librería hacía uso para Linkedin de la versión 1.0 de su API y, desde el 1 de mayo, Linkedin dejó de soportar dicha api, haciendo necesaria utilizar la versión 2.0 [16].

  Al haber tenido que dedicar gran parte del tiempo a implementar otras funcionalidades de la aplicación, y tener que compaginarlo con la terminación del 4º curso de grado, y

al contratiempo del cambio en la API producido hace menos de un mes, se ha decidido no implementar el login con Linkedin.

Como alternativa, se ha intentado utilizar **Auth0**, sin embargo, su funcionamiento es distinto, ya que no devuelve los datos a través de OAuth0, sino que la web auth0.com actúa como intermediario. Es decir, Capsule no realiza ni recibe los datos, los recibe Auth0.

Otras funcionalidades que no se han podido implementar por cuestión de tiempo son:

- Animaciones mientras el contenido está cargando.
- Mejorar los filtros de búsqueda, añadir filtro por texto y por red social.
- Editar las imágenes de una publicación programada.
- Realizar un apartado de estadísticas para ver las reacciones generales de una cuenta, no sólo publicación a publicación de forma individual; y cargar otros datos como el número de seguidores ganados o perdidos en un intervalo de tiempo.
- Añadir más consejos para mejorar la calidad de una publicación.

Tampoco se han implementado otros elementos definidos en el Lean Canvas referentes al flujo de ingresos: anuncios y un sistema de donaciones en Paypal, ya que, de momento, la aplicación se trata de un prototipo.

13.3. Conclusión y nociones aprendidas.

El objetivo de este proyecto no era desarrollar únicamente la plataforma web para ofrecer un producto y una serie de servicios. La intención era desarrollarlo todo desde 0; pasando por todas las fases, tanto del frontend como del backend.

Además, una de las partes en las que se ha hecho bastante hincapié ha sido, sobre todo, en el apartado del backend. Es una sección que, en Ingeniería Multimedia, no se profundiza mucho en comparación con otros relacionados con frontend y, por este motivo, además de desarrollar la aplicación web, quería aprender y aplicar cosas nuevas.

Gracias a este TFG, he configurado todo el servidor desde 0, en mi casa, en una Raspberry que está conectada al router, donde he tenido que abrir una serie de puertos, configurar una DNS dinámica, saber lo que es una CG-Nat (y solicitar a mi operador de internet salir de ella) y otras tareas como configurar en esa Raspberry una VPN de forma manual.

Estas tareas, perfectamente podrían no haberse realizado, pudiendo haber adquirido para el servidor cualquier VPS en plataformas como OVH. Del mismo modo, la VPN también sobraba, puesto que, para acceder al servidor, con SSH, la verificación en 2 pasos y herramientas como Fail2Ban, ya hacían bastante seguro el acceso al servidor. Sin embargo, son cosas que quería aprender. El acceder a una página web desde cualquier parte del mundo sabiendo que la misma se encuentra en el escritorio de mi habitación, o poder navegar desde cualquier sitio con la IP de mi casa, pudiendo acceder a la red interna de la misma y a los dispositivos, sin necesidad de estar allí presente, son cosas que me han llamado mucho la atención y me han gustado. Además, creo que me podrán ser muy útiles en un futuro.

Tareas como realizar scripts para programar publicaciones o envío de emails y generar copias de seguridad automáticas, junto con otras como crear todo el API Rest desde 0 creo que son muy importantes a la hora de desarrollar un producto, y que muchas veces, no siempre se tienen en cuenta o no se les da la importancia que deberían tener. Por este motivo, en el desarrollo de este proyecto, he querido centrarme tanto en el backend como en el frontend a partes iguales, aunque quizá, algo más en la parte del backend, puesto que era donde más cosas nuevas podía aprender.

Además, he aprendido bastante en todo lo relacionado con las APIs externas (para bien y para mal). Hoy en día, en un mundo prácticamente informatizado, donde prácticamente todo el mundo dispone de alguna red social; el haber entendido y saber manejar las mismas, ya no sólo para iniciar sesión, sino también para crear contenido u obtener datos de la misma, desde una aplicación creada por mí, ya es algo que me hace estar satisfecho con el trabajo realizado en este TFG.

Finalmente, en cuanto al frontend, he conocido mucho mejor el manejo sobre Angular. Al comienzo de este TFG, tenía muy pocos conocimientos sobre esta tecnología, pero a lo largo del desarrollo de este trabajo, he tenido que aprender muchísimas cosas y, si a esto le sumamos que en el segundo cuatrimestre en la carrera hemos tocado también bastante Angular, creo que ahora, tengo una base y un conocimiento sobre este framework bastante aceptable para enfrentarme a muchos retos del mundo laboral. No hay que olvidar que es uno de los frameworks más utilizados hoy en día y en muchísimas empresas buscan a gente con conocimientos en el mismo, y creo que este TFG es una buena prueba de ello.

14. Referencias

1. IAB Spain*, Estudio Anual de Redes Sociales, 2018*. Elaborado por Elogia. Disponible en: https://iabspain.es/wp-content/uploads/estudio-redes-sociales-2018_vreducida.pdf
2. García Durán, Alberto, *La comunicación empresarial y las redes sociales.* Publicidad y Relaciones Públicas Universidad Rey Juan Carlos. Disponible en: http://www.albertodeduran.es/wp-content/uploads/2014/08/1x07-La-comunicaci%C3%B3n-empresarial-y-las-redes-sociales.pdf
3. *Las 4 mejores redes sociales para empresas: Facebook, Twitter, Linkedin e Instagram*. Tecnología para los negocios, Cámara Valencia. Disponible en: https://ticnegocios.camaravalencia.com/servicios/tendencias/las-4-redes-sociales-fundamentales-en-la-empresa-facebook-twitter-linkedin-e-instagram/
4. Sixto Garcia, J. *Fundamentos de marketing digital*. Ed. Comunicacion Social Ediciones Y Publicaciones. 2016.
5. Núria Chinchilla, *Horario laboral y tiempo familiar de padres y madres en España*. Disponible en: https://blog.iese.edu/nuriachinchilla/2016/01/horario-laboral-y-tiempo-familiar-de-padres-y-madres-en-espana/
6. Europa Press, *Así se compara el horario laboral español con el de otros países europeos*, Disponible en: https://www.europapress.es/sociedad/noticia-asi-compara-horario-laboral-espanol-otros-paises-europeos-20161213135213.html
7. Olivares, Jorge, *¿Por qué el internet por cable se pone lento en la noche?,* Quora. Disponible en: https://es.quora.com/Por-qu%C3%A9-el-Internet-por-cable-se-pone-lento-en-la-noche
8. Moreno Molina, M. *Cómo triunfar en las redes sociales: Consejos prácticos y técnicas para conseguir todo lo que te propongas en Internet y sacarle más partido a tus redes ... Facebook, Twitter, Instagram, LinkedIn*. Ed. Gestión 2000, mayo 2015.
9. 27 of the Most Effective Social Media Marketing Tools You Need in 2019. Disponible en: https://shanebarker.com/blog/social-media-marketing-tools/
10. 20 Best Social Media Management Software Tools of 2019. Disponible en: https://financesonline.com/top-20-social-media-management-software-tools/
11. Alternatives to Hootsuite. Disponible en: https://alternativeto.net/software/hootsuite/
12. Libro de Estilos para la redacción de trabajos TFG/TFM de la EPS. Disponible en: https://maktub.eps.ua.es/servicios/gestorContenidos/contenidos/normativaEPS/Pdf/9910.pdf

13. El Cosmonauta. *Gmail, el correo más utilizado del mundo*. Disponible en: https://elcosmonauta.es/gmail-correo-mas-utilizado-del-mundo/
14. Post, retrive and engage with Tweets. Twitter Developers Program: https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/post-statuses-update
15. Post media/upload. Twitter Developers Program: https://developer.twitter.com/en/docs/media/upload-media/api-reference/post-media-upload
16. https://engineering.linkedin.com/blog/2018/12/developer-program-updates

15. Apéndice I

Creación de una VPN

Para poder acceder desde algún sitio que no pertenezca a la red interna, se ha configurado una VPN en la Raspberry de producción. Para ello se ha utilizado OpenVPN.

```
sudo apt-get install openvpn openssl
```

Una vez instalado OpenVPN, copiamos los scripts predefinidos easy-rsa en el directorio de configuración de OpenVPN:

```
sudo cp -r /usr/share/easy-rsa /etc/openvpn/easy-rsa
```

En el archivo **/etc/openvpn/easy-rsa/vars** encontramos algunos parámetros de configuración, es necesario cambiar **export EASY_RSA="`pwd`"** por **export EASY_RSA="/etc/openvpn/easy-rsa"** (u otra carpeta donde hayamos copiado en el paso anterior los scripts predefinidos easy-rsa). También podemos modificar la línea **export KEY_SIZE=2048**. Se trata del número de bits de la longitud de clave. Se puede cambiar a 1024 o 4098, pero en este caso se va a dejar en 2048, que es como viene por defecto.

Para guardar estos ajustes que hemos realizado, ejecutamos el script vars con el comando source, y cambiamos el nombre del archivo que generará por **openssl.cnf**:

```
cd /etc/openvpn/easy-rsa
sudo su
source vars
ln -s openssl-1.0.0.cnf openssl.cnf
```

Para crear las claves, primero creamos una para el servidor:

```
./build-key-server nombre_para_el_servidor
```

En caso de tener alguna clave antigua, primero sería necesario borrarlas todas para empezar de cero. Esto se hace con el comando:

```
./clean-all
./build-ca OpenVPN
```

Para crear las llaves de cada usuario, utilizamos el comando **build-key**. Podemos crear las llaves que queramos. Una vez las tenemos, generamos la clave Diffie-Hellman, que tardará un poco:

```
./build-key nombre_llave_1
…
./build-dh
exit
```

El siguiente paso es configurar los datos del servidor, el puerto que va a utiliza, el protocolo y qué IPs asignará a los clientes que se conecten y las DNS a utilizar. En este archivo también se definirá la configuración para los clientes que se conecten. Para ello, abrimos el archivo **/etc/openvpn/openvpn.conf** y configuramos lo siguiente:

```
dev tun
proto udp // PROTOCOLO QUE QUERAMOS UTILIZAR, EN ESTE CASO UDP
port 1234 //EL PUERTO QUE VAYAMOS A UTILIZAR

//Certificados necesarios
ca /etc/openvpn/easy-rsa/keys/ca.crt
cert /etc/openvpn/easy-rsa/keys/server.crt
key /etc/openvpn/easy-rsa/keys/server.key
dh /etc/openvpn/easy-rsa/keys/dh2048.pem

server 10.0.1.0 255.255.255.0 //Dirección IP
push "redirect-gateway def1 bypass-dhcp"
push "dhcp-option DNS 8.8.8.8" //Dns de Google
push "dhcp-option DNS 8.8.4.4"

//Para los clientes
persist-key
persist-tun
user nobody
group nogroup
status /var/log/openvpn-status.log
```

```
verb 3
client-to-client
comp-lzo
```

Una vez configurado, hay que activar las redirecciones para los paquetes con IPTables. Para ello se ha creado un script /etc/init.d/capsulevpn con la siguiente información:

```
#! /bin/sh
echo 'echo "1" > /proc/sys/net/ipv4/ip_forward' | sudo -s
iptables -A INPUT -i tun+ -j ACCEPT
iptables -A FORWARD -i tun+ -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -F POSTROUTING
iptables -t nat -A POSTROUTING -s 10.0.1.0/24 -o eth0 -j MASQUERADE
```

Damos permisos de ejecución al script y lo ejecutamos:

```
sudo chmod +x /etc/init.d/capsulevpn
sudo update-rc.d capsulevpn defaults
sudo /etc/init.d/capsulevpn
sudo /etc/init.d/openvpn restart
```

Para la configuración del cliente, podemos crear un archivo **.ovpn**, con todos los datos de la configuración. De esta manera, luego es mucho más fácil conectarse. En este archivo se especifica la dirección de la VPN, el puerto, el protocolo y los certificados a utilizar:

```
dev tun
client
proto udp //Protocolo
remote capsule.onthewifi.com 1234 //direccion y puerto
resolv-retry infinite
nobind
persist-key
persist-tun

//Los certificados
ca ca.crt
```

```
cert nombre_llave_1.crt
key nombre_llave_1.key

comp-lzo
verb 3
```

Los certificados necesarios (ca.crt, *.crt y *.key) se crearon en la carpeta **/etc/openvpn/easy-rsa**, donde creamos las llaves para cada usuario. Cada usuario tendrá un .crt y un .key con el nombre que se estableció al crear la llave.

Finalmente, necesitamos estos 4 archivos: el **.ovpn** que hemos creado con la configuración, el **ca.crt**, el **.crt** y el **.key** con el nombre de cada llave.

Con estos 4 archivos, ya podemos conectarnos a la VPN a través de OpenVPN. OpenVPN se puede utilizar tanto en Windows, Linux como MacOS, así como en Android o iOS. Bastaría con seleccionar el archivo **.ovpn** que hemos creado donde se encuentra la configuración, y OpenVPN se encargaría de lo demás.

También se podría hacer sin OpenVPN, pero tendríamos que configurar la VPN de forma manual, con el **ca.crt,** el **.crt** y el **.key**, así como la url, el puerto y el protocolo. OpenVPN simplemente facilita la conexión.

Para que funcione con OpenVPN es necesario, eso sí, que los 4 archivos se encuentren en el mismo directorio.

Para **Linux**, el comando a ejecutar es el siguiente:

sudo openvpn –config capsule.ovpn

Si no tenemos openvpn instalado, es necesario realizarlo antes. Para ello, lo hacemos con el comando:

sudo apt-get install openvpn

En las siguientes figuras, podemos observar cómo la VPN está funcionando tanto en un ordenador Linux como en un smartphone Android:

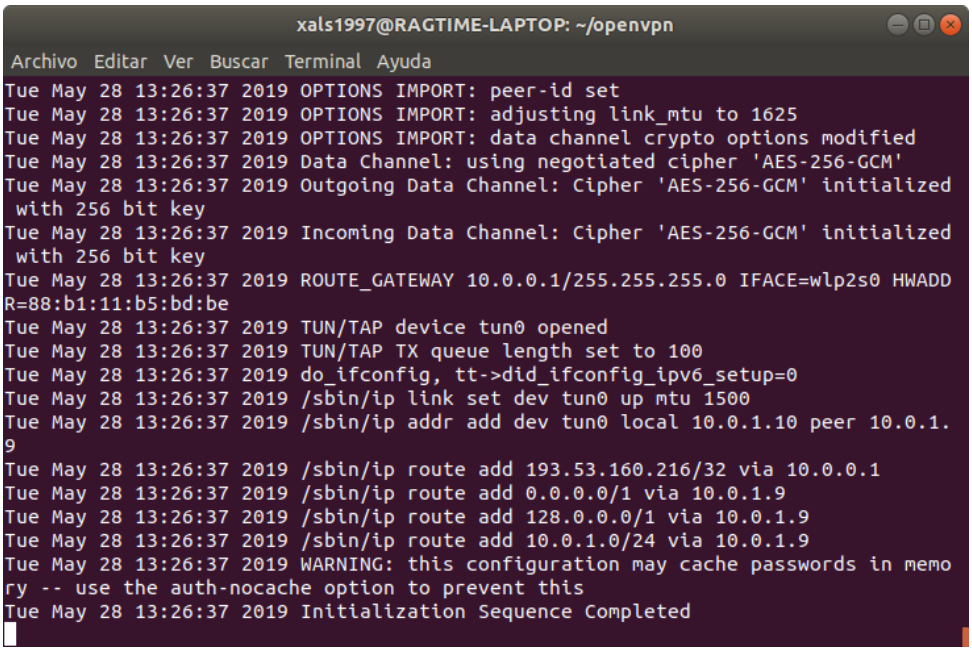


Figura 72. Navegando a través de la VPN desde Linux. Como vemos, se ha asignado la IP 10.0.1.10.

Probando la VPN desde Android:

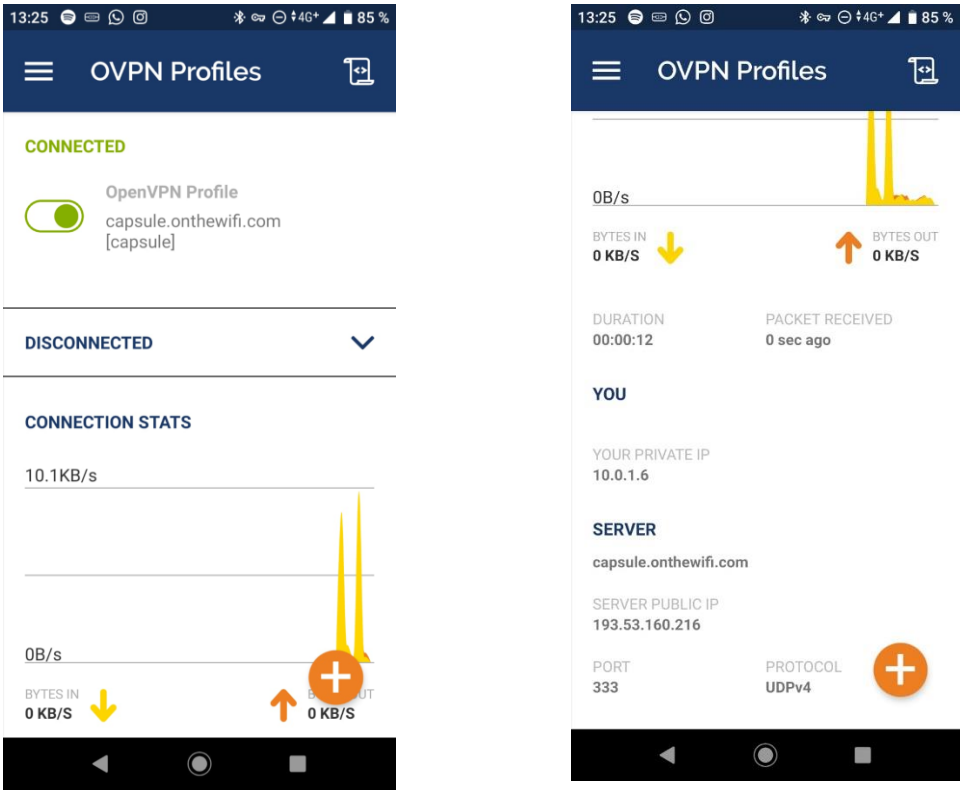


Figura 73. OpenVPN desde Android.

Como vemos en las capturas de Android, estamos conectados a la red 4G, por tanto, el dispositivo no se encuentra en la red interna. Al conectarnos a la VPN, en este caso, se nos ha asignado la IP interna 10.0.0.6, y estamos navegando por internet desde la IP pública del router de mi casa en este momento, y no desde la IP que me haya asignado mi operador de telefonía móvil.